Real-Time Markerless Tracking the Human Hands for 3D Interaction

Dissertation

zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Dipl.-Inf. Markus Schlattmann
aus
Aachen

Bonn, April 2010

Universität Bonn
Institut für Informatik II
Römerstraße 164, D-53117 Bonn

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen
Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Reinhard Klein

2. Gutachter: Prof. Dr. Gabriel Zachmann

Tag der Promotion: 17.12.2010
Erscheinungsjahr 2011

CONTENTS

List of Videos

Several videos are available either on the accompanying CD (print version) or on the thesis download server (electronic version). Each video is titled using the number as well as the headline of the according thesis section. If more than one video belongs to one section an additional comment is appended to the end of the filename. The following videos are available:

4 Tracking Two Hands.wmv

7.4 Hybrid Cursor Control - Bare-Handed.mov

7.4 Hybrid Cursor Control - Nintendo Wii-Remote.mov

10.1 Virtual Building Blocks.wmv

10.2 Mesh Editing - Deformation.wmv

10.3 Virtual Pointer.mov

10.4 3D Manipulation Interface - One-Handed.wmv

10.4 3D Manipulation Interface - Two-Handed.wmv

11.1 Zugspitze 3D (Fly Through).wmv

11.2 RTT-DeltaGen (3D Manipulation).mpg

11.3.1 Racing Simulator.wmv

11.3.2 Flight Simulator.wmv

11.3.3 First-Person Shooter.wmv

List of Figures

LIST OF TABLES

ABSTRACT

This thesis presents methods for enabling suitable human computer interaction using only movements of the bare human hands in free space. This kind of interaction is natural and intuitive, particularly because actions familiar to our everyday life can be reflected. Furthermore, the input is contact-free which is of great advantage e.g. in medical applications due to hygiene factors.

For enabling the translation of hand movements to control signals an automatic method for tracking the pose and/or posture of the hand is needed. In this context the simultaneous recognition of both hands is desirable to allow for more natural input. The first contribution of this thesis is a novel video-based method for real-time detection of the positions and orientations of both bare human hands in four different predefined postures, respectively. Based on such a system novel interaction interfaces can be developed. However, the design of such interfaces is a non-trivial task. Additionally, the development of novel interaction techniques is often mandatory in order to enable the design of efficient and easily operable interfaces. To this end, several novel interaction techniques are presented and investigated in this thesis, which solve existing problems and substantially improve the applicability of such a new device. These techniques are not restricted to this input instrument and can also be employed to improve the handling of other interaction devices. Finally, several new interaction interfaces are described and analyzed to demonstrate possible applications in specific interaction scenarios.

ZUSAMMENFASSUNG

In der vorliegenden Arbeit werden Verfahren dargestellt, die sinnvolle Mensch-Maschine-Interaktionen nur durch Bewegungen der bloßen Hände in freiem Raum ermöglichen. Solche "natürlichen" Interaktionen haben den besonderen Vorteil, dass alltägliche und vertraute Handlungen in die virtuelle Umgebung übertragen werden können. Außerdem werden auf diese Art berührungslose Eingaben ermöglicht, nützlich z.B. wegen hygienischer Aspekte im medizinischen Bereich.

Um Handbewegungen in Steuersignale umsetzen zu können, ist zunächst ein automatisches Verfahren zur Erkennung der Lage und/oder der Art der mit der Hand gebildeten Geste notwendig. Dabei ist die gleichzeitige Erfassung beider Hände wünschenswert, um die Eingaben möglichst natürlich gestalten zu können. Der erste Beitrag dieser Arbeit besteht aus einer neuen videobasierten Methode zur unmittelbaren Erkennung der Positionen und Orientierungen beider Hände in jeweils vier verschiedenen, vordefinierten Gesten. Basierend auf einem solchen Verfahren können neuartige Interaktionsschnittstellen entwickelt werden. Allerdings ist die Ausgestaltung solcher Schnittstellen keinesfalls trivial. Im Gegenteil ist bei einer neuen Art der Interaktion meist sogar die Entwicklung neuer Interaktionstechniken erforderlich, damit überhaupt effiziente und gut bedienbare Schnittstellen konzipiert werden können. Aus diesem Grund wurden in dieser Arbeit einige neue Interaktionstechniken entwickelt und untersucht, die vorhandene Probleme beheben und die Anwendbarkeit eines solchen Eingabeinstruments für bestimmte Arten der Interaktion verbessern oder überhaupt erst ermöglichen. Diese Techniken sind nicht auf dieses Eingabeinstrument beschränkt und können durchaus auch die Handhabung anderer Eingabegeräte verbessern. Des Weiteren werden mehrere neue Interaktionsschnittstellen präsentiert, die den möglichen Einsatz bloßhändiger Interaktion in verschiedenen, typischen Anwendungsgebieten veranschaulichen.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Dr. Reinhard Klein, for all his help, support and guidance. It was him, who motivated and prepared me for the field of computer graphics and this way made this work possible.

I want to thank Prof. Dr. Gabriel Zachmann for being the second reviewer and his valuable suggestions and comments. For kindly agreeing to be the other members of the audit committee I also want to express my gratitude to Prof. Dr. Stefan Wrobel and Prof. Dr. Nikolaus Froitzheim.

My thanks go to all my former colleagues at the computer graphics group at the University of Bonn. I always enjoyed the very friendly atmosphere, interesting discussions at lunchtime and substantial support for my work. In particular, I want to thank Ferenc Kahlesz for our joint work and the possibility to use his great calibration and camera grabbing implementations. I am also very grateful for the valuable feedback he had given to me during my research. Special thanks go also to Ralf Sarlette, who took care of all the hardware issues, and Alexander Greß for his support in GPU-programming. Additionally, I want to thank Ruwen Schnabel, Alexander Greß, Gero Müller, Patrick Degener, Raoul Wessel, Ákos Balázs and Michael Guthe for proofreading and further help on preparing publications. I also want to mention the great time I had sharing an office with Gero Müller, Ruwen Schnabel, Alexander Greß and Max Hermann.

I am very grateful for the valuable work of Roland Ruiters, Tanin Na Nakorn and Johannes Broekelschen, who temporarily worked for me as student assistants. Thanks go also to Tan Zheng for the work during his master thesis.

A big thank belongs to all of the participants in the diverse user experiments. I am very grateful for the patience they had to finish the presented tasks.

At last but by far not least I want to thank my family for all their support over the years. You gave me the strength I needed to finish this work.

CHAPTER 1

INTRODUCTION

1.1 Hand-Tracking

Ever since the creation of the first Virtual/Augmented-Reality (VR/AR) applications Human-Computer-Interaction research has taken a great interest in using the human hand as an input-device. This is explained by one of the main requirements of VR/AR: actions/responses familiar from our everyday life have to be at the user's disposal, and the overall user experience should possibly be enhanced by allowing the user to carry out tasks, which are impossible in real life (e.g. flying). Everyday interaction with our environment involves our hands as natural "interaction-devices", therefore, it should also be available in its virtual counterpart.

To enable VR/AR systems to react to hand movements, the movement of the user's hand has to be tracked. Thereby several tracking requirements are essential in order to support immersive user experience. Two requirements in particular should be fulfilled: first, the tracking should be done in real-time, because otherwise instant interaction would be impossible. Second, no extra devices should be needed apart from the user's hand (e.g. datagloves or markers) to ensure immersive interaction. Furthermore it is desired that the initialization and detection if the previous tracking was correct are handled fully automatically, because otherwise the user constantly would have to verify by herself/himself, whether her/his actions are interpreted appropriately. This would prohibit the user from concentrating on the actual task that she/he tries to carry out. Another property that would reduce the general availability of a system for a potential new user is the avoidance of time-consuming per user calibration and/or registration procedures.

Having defined the requirements for hand-tracking systems in general, we can turn our attention to the question: what features should actually be tracked? The human hand can be modeled satisfyingly as a kinematic chain with 26-27 degrees of freedom: 6 DOF for the pose (global position and orientation) of the hand and 20-21 for the joint angles of the fingers [RK95, KHW95]. The extent people

exploit the full DOFs of their hands and whether there is an additional temporal context to the hand movement (dynamic gestures, e.g. waving) strongly depends on the intention of one's action. Thus, whether all DOFs or only a subset has to be tracked depends on what kinds of interaction the system should support.

For example in a dexterous manipulation simulation it is essential to track all DOFs of the hand. Unfortunately, no visual tracking approach exists that is capable of tracking all DOFs or even more than 6 DOFs and several stiff postures (specified configuration of finger limb positions and orientations relative to the hand pose, e.g. the fist posture) of the hand in real-time while using affordable hardware. Note that by tracking the hand pose over time also arbitrary hand gestures (predefined movements of the hand pose, e.g. writing a letter in the air) can be recognized and exploited.

In order to further increase the intuitiveness of such interaction interfaces, the system should also be capable of simultaneously tracking both human hands. Kotranza et al. [KQL06] performed a user study concerning this matter with the result, that most people intuitively use both hands for simulated hands-on tasks and that two-handed virtual interaction better resembles real-world tasks. Several other 3D applications use two-handed interaction for a more intuitive handling.

1.2 Interaction Techniques

While markerless hand-tracking enables the translation of real hand movements to virtual movements, several problems have to be solved in order to build a 3D interface on top of the basic hand-tracking technology. Exploiting the detection of position, orientation and posture the interface has to provide mechanisms for basic interaction techniques such as object selection, grabbing/releasing and 3D navigation. Thereby an easy to use realization of these techniques is crucial for usability and efficiency of the interface. These techniques have to be adapted to the users' capabilities and incapabilities (e.g. the limited range of angle movements of the human wrist), the applications' specifications as well as the requirements and drawbacks of the used hand-tracking method. This diversity of demands poses several challenges in the design of easy to use interfaces as described in the following.

The first challenge is grabbing and releasing of objects, which are inherent tasks during 3D interaction sessions due to the following reasons. The virtual world is typically significantly larger than the working volume of the user (the 3D region the hand is tracked in). Therefore, to enable users to move a virtual object to every position in the virtual space it has to be possible to grab it and release it in order to move it step by step. Scaling the working volume to the whole virtual world is not an option, because the accuracy would decrease too heavily. Similar,

the range of angle movements of the human wrist is limited. In order to fully rotate and inspect an object from all viewing directions grabbing and releasing are indispensable.

In interfaces that employ a standard 2D mouse, grabbing and releasing is solved either by lifting the mouse (while it is lifted a mouse movement induces no object movement) or by exploiting button states (usually the object is only moved if a button is held down). But in contrast to standard controllers no direct adequate exists for markerless hand-tracking. Simple solutions for the realization of a grab and release cycle in the absence of physical buttons are disposing one degree of freedom (DOF) of the hand pose, e.g. only if z-coordinate is greater than a certain value the object is grabbed, exploiting the second hand or applying different postures for different button states. Unfortunately, these approaches also have drawbacks. Exploiting one DOF of the hand pose is only possible, if less than 6 DOF are needed in the current task. Using the second hand to define the current grab and release state is more uncomfortable due to the need of straining both hands and arms simultaneously and therefore should not be used as a permanent mode of action. However, in specific scenarios temporarily switching to two-handed control can significantly improve certain manipulation tasks (see e.g. [OKF+05]). The use of different postures, one for grabbing and another for releasing, is significantly more demanding for the user than simply pressing a mouse button, because the physical effort as well as the complexity of coordination of changing postures is considerably higher, especially if various specified postures are concurrently needed. Moreover, a posture change always induces an unintended pose change mainly in rotation in current markerless hand-tracking systems. This is due to the problem that the tracking state is temporarily undefined during a posture change. Therefore, it would nearly be impossible to instantly stop an object's movement by switching to another posture.

The second challenge is the lack of suitable techniques for selecting objects and tools. In standard interfaces typically selection is performed by moving the cursor above a virtual button or object on the screen and performing a click. To this end, cursor positioning as well as clicking simulation has to be suitably solved in the context of bare-handed interaction. Unfortunately, free-hand cursor positioning lacks precision due to natural hand tremor and tracking inaccuracies. Increasing precision with a simple speed dependent acceleration scheme is not straightforward for bare-handed interaction without affecting the intuitiveness and effectiveness. Because of the lack of physical buttons enabling bare-handed clicking is also challenging and not yet solved suitably.

Another challenge of using hand-tracking for interaction is providing the user information about the limited working volume of the hand-tracking device and details about the current tracking state. The user has to ensure permanently that her/his hand is located inside the working volume and that she/he is performing the

correct pose and posture for solving the current task. Without suitable techniques to facilitate these needs, this can lead to extremely high demands for the user.

Last but not least, in a 3D interface comprising several different interaction modes (e.g. selection and manipulation) switching between these modes has to be enabled. Therefore, suitable mechanisms are needed which minimize the additional effort, complexity and disturbance for the user.

1.3 Interaction Interfaces

Having a system to real-time track hand movements, suitable interfaces are needed to fully exploit its capability for efficient and intuitive interaction. Adequate interaction techniques/metaphors have to be selected, combined and adapted to fit the specifications of an interaction scenario.

In the past decades numerous interaction interfaces for industrial applications were introduced such as assembly simulation or handling virtual menus. However, most of these interfaces are designed for being used with other interaction devices (e.g. data gloves or physical controllers) and therefore employ interaction techniques that are not suitable for our case. Moreover, previously proposed interfaces obviously can only consider interaction techniques being introduced before. Hence, these interfaces did not consider novel techniques which could significantly improve certain interaction tasks. Because of these facts, previous interaction interfaces still suffer from several drawbacks concerning their use for bare-handed interaction. For example, compensating the absence of physical buttons or enabling precise and fast manipulations with a limited working volume are challenges that were not sufficiently so far.

In the field of bare-handed interaction interfaces for games only little research was performed in the last decades. In the last years research in this special field gained more interest, also because of the great success of novel interaction devices as for example the Nintendo Wii-Remote controller. But the Wii-Remote includes the need of holding a physical device and is not able to fully recover the global pose. In contrary an adequate markerless vision-based hand-tracking system does not suffer from these drawbacks, but needs further investigation of suitable interaction metaphors for gaming. Moreover, user experiments are needed to analyze the overall applicability of bare-handed tracking for game control.

1.4 Goals

The primary goal of this thesis is the development of a novel interaction device that enables real-time bare-hand user input and is capable of detecting both global

hand positions and orientations without angle limitations. In order to exploit the full potential of this device an inherent objective is designing suitable interaction techniques and interfaces.

The first step toward this end is developing a robust hand-tracking system with the according capabilities. Having such a system it is possible to implement interaction interfaces for closing the gap between the device and possible applications. As the design of such an interface is crucial for usability and acceptance, careful investigation of possible ways of interaction is the obligatory second step. This comprises the identification and adjustment of existing as well as the development and experimental analysis of novel interaction techniques. Moreover, suitable combinations of interaction techniques have to be identified and integrated into exemplary interfaces to demonstrate the use of the device in typical interaction scenarios.

1.5 Main Contributions and Thesis Structure

The main content of this thesis is organized in the following three parts:

Part I - Hand-Tracking: In this part our work on a system for tracking the bare human hands is presented. A visual hand-tracking system was designed and implemented that fully satisfies the requirements mentioned in Sec. 1.1. This system is capable of tracking the global 6 degrees of freedom (position and orientation) of one user hand in 4 predefined postures. Subsequently, this system was extended for the simultaneous tracking of both user hands. This work forms a basis for our research in bare-handed interaction.

Part II - Interaction Techniques: In this part several interaction techniques are presented, which can heavily support bare-handed human computer interaction as well as interaction with other free-hand devices. The following central problems are addressed: one and two-handed virtual 3D object manipulation without physical buttons, menu navigation and clicking with bare hands, compensating the missing force feedback by visual feedback. Additionally, the results of several user experiments are presented which show the efficiency of several of these techniques.

Part III - Interaction Interfaces and Applications: Several self-developed interaction interfaces and applications as well as suitable interfaces to commercial applications are presented and investigated in this part. Among others, solutions to 3D object manipulation, virtual pointer control, virtual fly through and 3D game control are introduced. In the context of bare-handed gaming as well a user study is presented.

As usual in the field of computer graphics and human computer interaction most of the algorithms presented in this thesis have already been published or submitted as international conference/journal papers [SKSK07, SK07, SNNK09, SK09a, SBK09, SK09c, SK10, SZBK10], patent specifications [SK09d, SK09b] and a master thesis [Zhe09].

Part I

Hand-Tracking

CHAPTER 2

PREREQUISITES

In this chapter several basic techniques and procedures needed for our hand-tracking method are discussed. Our method relies on a coarse 3D-reconstruction of the user's hand, which is obtained by constructing the so called visual hull. This concept is outlined in the first section of this chapter. The visual hull construction technique requires multiple calibrated cameras and a well suited segmentation procedure for segmenting the camera images into hand and non-hand pixels. To this end in the following two sections a suitable calibration method is outlined and it is explained how segmentation is performed. Subsequently, a novel procedure for real-time analysis of the segmentations is introduced. In the last section of this chapter the hardware prototypes of our hand-tracking system are described.

2.1 The Visual Hull

The Visual Hull is defined as the geometric entity created by the shape from silhouettes 3D reconstruction technique introduced by Laurentini [Lau94]. This technique requires multiple 2D silhouettes of a 3D object from different views. Along with the viewing parameters of the respective camera, a 2D silhouette defines a back-projected generalized cone that contains the 3D object. The intersection of multiple cones defines a bounding volume of the object and is called visual hull. An illustration is given in Fig. 2.1. The visual hull provides a coarse 3D volume representation of the object. Thereby, the quality of the 3D representation depends strongly on the arrangement of the cameras. The best average quality is achieved if the directions the object is viewed from uniformly sample the unit half sphere.

In our hand-tracking system we compute the visual hull of a hand in each frame (i.e. for each new set of camera images) and then extract and identify features in the 3D domain in order to determine the global pose and posture of the hand. To this end, all important parts of the hand (i.e. hand parts comprising the features) have to be located in the region that is captured from all cameras. We refer to this region as the working volume. The working volume is described by the visual hull computed by using the full image rectangles as silhouettes (see Fig.

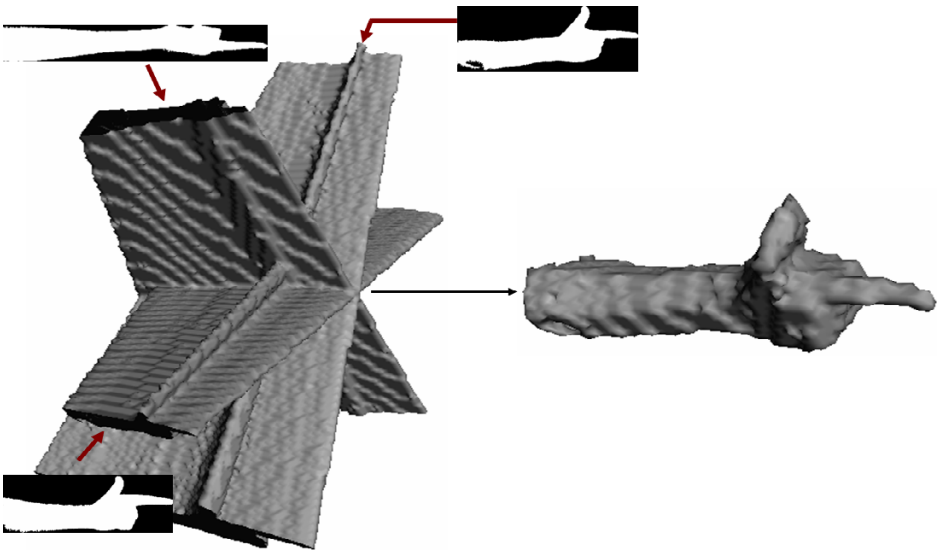


Figure 2.1: Construction of the visual hull of a hand from three silhouettes.

2.2).

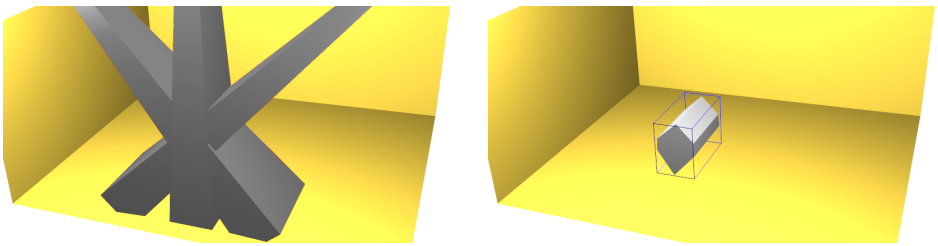


Figure 2.2: The visual hull computed by using the image rectangles as silhouettes. The resulting convex polytope (Right) is defined to be the working volume. Note that the visual hull of any image segmentations is always contained in the working volume.

2.2 Camera Calibration

In order to allow for constructing the visual hull, the intrinsic and extrinsic parameters of the cameras have to be calibrated. The intrinsic parameters determine the optical, geometric and digital characteristics of the camera. They can be described by the perspective projection, the transformation between image plane coordinates and pixel coordinates and the geometric distortion introduced by the bending of the lens. These parameters can be obtained separately for each camera.

The extrinsic camera parameters uniquely identify the transformation between the camera reference frame and the world reference frame. Determining these parameters means identifying the camera positions and orientations in one global coordinate system. Obviously, this can not be done separately for each camera.

Camera calibration is usually done by showing an object with known geometry and color distribution to the cameras which provides visual feature points that can unambiguously be identified in the camera images. As the relative feature positions are known in the global reference frame the computed feature positions can be used as control points to estimate the intrinsic camera parameters. Furthermore, identifying corresponding feature positions for multiple cameras enables the estimation of the extrinsic camera parameters.

For determining the camera parameters we adopt the calibration method described in [KLK07]. We chose this method mainly because the triangulation error is explicitly taken into account, thus making the calibration results well suited for the shape-from-silhouettes technique. This method consists of the following steps:

Data Acquisition: The user has to show a planar checkerboard pattern to the cameras; however, it does not have to be visible by all the cameras simultaneously. While the user moves the pattern, the synchronized video streams of the cameras are captured.

Feature Extraction: The predefined features of the pattern (the chessboard corners) are extracted from each captured camera image.

Distortion Estimation: The lens distortion parameters are calibrated for each camera. Subsequently, the dataset (i.e. images and features) is undistorted.

Intrinsic Calibration: The intrinsic parameters (i.e. focal length and the transformation between image plane coordinates and pixel coordinates) are identified independently for each camera.

Extrinsic Calibration: The extrinsic parameters (i.e. positions and orientations) of the cameras are determined by exploiting the inherent feature correspondences.

Although the need for a planar calibration object makes the method slightly more inconvenient for the user than e.g. [SMP05], it allows to decouple the full pinhole calibration into three subproblems with smaller dimensionality. This also has the benefit that the feature extraction, distortion estimation and intrinsic calibration can be parallelized.

2.3 Image Segmentation

In order to compute the visual hull, the camera images need to be segmented to hand/arm and non-hand/arm pixels at each frame. The quality of the hand reconstruction using the visual hull directly depends on the quality of the segmentation. As the further processing stages of our hand-tracking build upon the visual hull, it is important to use well-suited mask images to build it. Robust real-time segmentation in general environments is an open problem, therefore we decided to control the surroundings in the working volume enabling a simple background subtraction algorithm for segmentation. In a background subtraction algorithm a previously learned background image (typically obtained by averaging several images before the user's hands are inside the working volume) is subtracted from the current camera image and the segmentation is applied to the resulting difference image. If the foreground (i.e. the user's hands and arms) differs strong enough from the background a simple thresholding on the differences is sufficient for segmenting very reliably. In the following the process of updating the background is explained.

2.3.1 Background Update

Updating the background image means replacing the color/intensity of the background image by the color/intensity of the current image in a certain way. The simplest solution is replacing the background image with the current image. However, due to aliasing artifacts and small errors in the image capturing process, it is superior to replace the background image with the average of a sequence of images. Therefore, typically n images are accumulated in an image buffer and each pixel value is divided by n to get the background image. In our case, we want to successively add the color/intensity information of single images to the background buffer. To this end, in a first step the accumulation buffer is initialized by summing up a sequence of n images. Then, the new image is added to the accumulation buffer and the oldest image that contributes to the accumulation buffer (i.e. was added and not yet subtracted) is subtracted. After that, the new background is computed by dividing by n. In our setting we used $n = 25$. If only specified regions of the background shall be updated, the accumulation can be performed pixel-wise; pixels of the background belonging to such a specified region are updated individually.

In our system, if it is known that currently no hand/arm is located inside the working volume (WV), the update routine is always running for the entire images. In the case that a hand is decided to be located inside the WV, the update routine is either only partly (i.e. for specific image regions) or not running. The decision if the WV is empty and which regions can be updated is made by an automatic

segmentation analysis described in Sec. 2.4.

2.4 Real-Time Segmentation Control

Under the assumptions that each camera background is static and contains no items having a similar color (if color cameras are used) or intensity (if infrared cameras are used) as the foreground (hands/arms), a background subtraction algorithm segments very reliably. However, changes in the environment (e.g. lighting conditions, adding/removal of objects) can lead to problematic changes of the background. In practice such changes can not always be avoided, particularly, if the system is used for a longer period of time without supervision. Mainly two factors lead to problematic background changes: varying illumination and changes by the user. The illumination can vary slowly (e.g. solar altitude) or abruptly (e.g. switching lights on/off). In both cases the background has to be updated to prevent incorrect segmentation. The challenge is adding only the changes of the background to the background buffer. Falsely regarding foreground as background has to be avoided. If the current segmentation can be guaranteed to be correct, it is possible to update only the image region belonging to the background. However, this can not be ensured if the illumination changes abruptly. In this case, the background should only be updated if the working volume is empty.

If the user herself/himself changes the background by adding/removing items to/from the sensitive region (e.g. a self brough item or a body part not used for interaction) the segmentation can seriously be disturbed. In this context we distinguish two kinds of items: similar or not similar to the foreground. If the background contains items too similar to the foreground (e.g. a body part), it is impossible to perform correct segmentation in the respective image region. Therefore, this case has to be detected to enable instructing the user to remove such items. If the items are not similar to the foreground, segmentation can work correctly if the background is updated. To this end, this case has to be detected and a certain update process has to be performed.

2.4.1 Automatic Segmentation Analysis

The idea is to establish a context between the different camera images for gaining knowledge about the correctness of the segmentation. We exploit the fact that more than two cameras are concurrently used and their spatial context is known.

Exploitation of Spatial Context

In order to meaningfully compare the segmentations of all camera images, we first construct the visual hull of the image segmentations. Then, we project the visual hull to each segmentation image (see Fig. 2.3) and inspect how the visual hull projections are overlapping the segmentations. The overlaps provide information

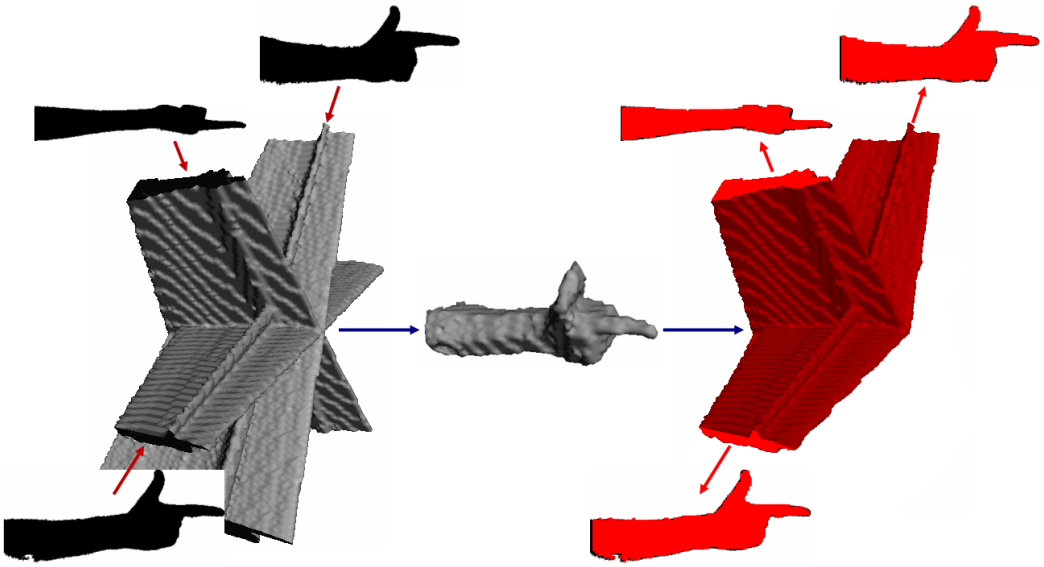


Figure 2.3: Illustration of the visual hull construction and projection process. First, the silhouettes (black hand/arm masks) are back-projected for the construction of the visual hull (see also Sec. 2.1). Then, the visual hull is projected to the images (red hand/arm masks). Note that by definition the projection of the visual hull is always contained in the image segmentations.

about the correctness of the segmentations as described in the following. If all images are segmented correctly, the true 3D object (i.e. hand/arm) is contained in the visual hull. Therefore, the visual hull projections to the images approximately coincide with the segmentations. An incorrect segmentation means that the segmentation either missed parts of the silhouette or contained parts not belonging to the silhouette of the true 3D object.

In the first case, if the segmentation in one image lacks some significant parts of the silhouette, these parts are also missing in the visual hull and in its projections. Therefore, depending on the viewpoints, at least one projection to another image would probably not coincide with its segmentation (see Fig. 2.4).

If the segmentation of an image contains significant parts not belonging to the silhouette, these parts are at least partially truncated from the visual hull as

Figure 2.4: In the segmentation on the right a part of the hand/arm is not contained in the foreground. Therefore, the projections of the visual hull to the other images (left and middle) does not coincide with their segmentations.

they are not contained in the segmentations of the other images. Therefore, the projection to this image is only partially overlapping its segmentation (see Fig. 2.5).

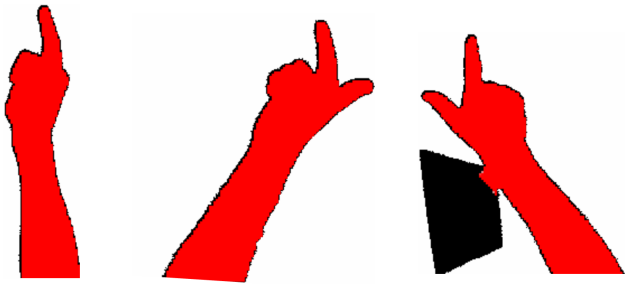


Figure 2.5: In the segmentation on the right a part of the background is contained in the segmentation. As it is not contained in the other segmentations (left and middle), it is truncated from the visual hull and not contained in the projection to the image.

To decide whether the overlap in an image is full (the projection nearly fills in the complete segmentation) or partial (the projection area is significantly smaller than the segmentation area), we use a certain threshold (95% of the segmentation area).

Analyzing the entire overlaps of visual hull projections and segmentations works fine if the cameras are capturing the same 3D space. However, this is typically not the case, because the cameras have very different positions and orientations. Therefore, the projection of the visual hull can miss a significant part of an image's segmentation even if all camera images are segmented correctly. This happens if the corresponding part of the hand/arm is truncated in another image by the image border (see Fig. 2.6). That leads to truncation of the visual

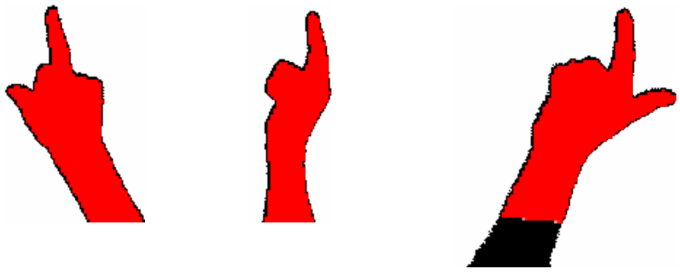


Figure 2.6: Although all images were segmented correctly, the visual hull projection on the right does not coincide with the segmentation. This happens when a segmentation in another image was truncated by the image border.

hull and its projection. To solve this problem, these regions have to be identified and ignored in the computation of the overlaps of segmentation and visual hull projection. Fortunately, these regions can be identified without user interaction. Our solution exploits the fact that the visual hull of the entire images (defined as the working volume, see Sec. 2.1) defines the 3D space where every 3D point has a corresponding 2D projection point in each camera image. 3D points not being located inside the working volume lack a corresponding 2D projection point in at least one camera image (i.e. the projection point is located beyond the image border). Therefore, the projections of the working volume can be used to classify the images. We refer to image pixels overlapping the projection of the working volume as active image pixels. The remaining image pixels are inactive and ignored in the segmentation analysis and update process. Note that the working volume and its projections to the images depend only on the camera parameters. Therefore, this method can be executed off-line in a pre-processing step.

Implementation

To maintain a correct segmentation over time without supervision, our method distinguishes the following three cases leading to different mechanisms:

Minor segmentation area in one image: If not enough active pixels in an image belong to the segmentation, the hand/arm is probably not located inside the working volume.

This is generally used as an indicator for an empty working volume. Therefore, the background update routine starts automatically (see. Sec. 2.3.1). This way, slow illumination changes as well as partial changes of the background are handled.

Partial overlap in one or more images: The segmentation is probably incorrect

in at least one image. Induced either by having added/removed items or abrupt illumination changes.

A message is given (acoustically and/or visually) to the user to remove any items and her/his body from the working area. Then, the background update routine is started. As soon as the background is fully updated and the segmentation areas are below a certain threshold, the user is informed to be able to continue her/his work.

Full overlap in all images: The segmentations are probably correct. The segmentations of the images correspond to each other, therefore the projections of the visual hull nearly coincide with the segmentations (see Fig. 2.3).

In this case, it is possible to use the projections of the visual hull as specified regions for partially updating the background images. Therefore, all active background buffer pixels not belonging to the projections of the visual hull are updated. This way, the segmentation is improved without disturbing the user. Like in the first case (see above), slow illumination changes and partial background changes are handled. To account for sampling errors, we first apply several dilatation steps to each visual hull projection mask.

By using this kind of segmentation analysis and handling, the system can run without supervision. Slow illumination changes are handled by continuously updating the background image buffers. If the illumination changes abruptly, the user is instructed to remove her/his body from the working volume and the background is updated. If needed, the user is also reminded to remove distracting objects/body parts from the working area. Fortunately, abrupt illumination changes typically happen infrequently and the user normally quickly learns not to put down distracting objects in the sensitive region. Therefore, breaking the work flow for relearning the entire background images is needed only rarely.

Note that this kind of segmentation analysis and control can also be used for improving other segmentation methods as for example skin color segmentation. Having learned an initial skin color histogram this histogram could successively be updated using the information about the overlap of segmentation and visual hull projection.

To implement the visual hull projection in real-time, first, the visual hull is computed on the GPU according to [LMS03] with a grid resolution of maximal $128 \times 128 \times 128$. Depending on the camera resolution, lower visual hull resolutions can also be sufficient. The information about whether a voxel is filled or not filled is stored in a binary format in the texture grid (we used a char texture and packed eight voxels to one char value). Therefore, the texture needs a maximal memory space of 262144 bytes (128^3 bits). Then, the texture is loaded to CPU memory and the contour of the visual hull (the contour contains all filled voxels,

which have filled as well as empty neighbors) is extracted. The contour typically contains significantly less voxels than the full grid (about 128^2 contour voxels if 128^3 voxels in total). Therefore, the subsequent projection to the segmentation images can be performed very fast.

On prevalent hardware (e.g. Intel Core 2 Duo 6600, Geforce 8800 GTX) the run-time of the visual hull construction, download to CPU memory and computation of the visual hull contour need less than 6 milliseconds and the projection to one image can be performed in less than one millisecond. Note that if the tracking algorithm anyway needs to compute the visual hull, the extra cost of the segmentation control is further reduced.

2.5 Hardware Setups

During our research we established three different hardware prototypes for tracking the human hands, two high quality active infrared night vision camera setups and one low cost webcam setup. In each setup the user's hands' movements are captured from above by three cameras and both systems adopt background subtraction for image segmentation. The cameras are connected to one computer (Intel Core 2 Quad Q8200, Geforce GTX 280), respectively, which is responsible for running our hand-tracking procedure.

Choosing background subtraction for segmentation confines the placement of the cameras, because segmentation is not possible if body parts other than arms and hands are observed. Therefore, the cameras must not be placed in front or behind the user. The cameras may only be positioned nearly circularly in parallel to the user's upper body around the working volume in front of the user. Such a placement of the cameras is clearly suboptimal for the visual hull computation; however, in practice we found that reliable segmentation is a lot more important than optimal or near-optimal camera placement for the visual hull computation. Given this restrictions, the angle between viewing directions of two adjacent cameras in our setups is approximately $60°$ illustrated in Fig. 2.7. Note that the reconstruction results can additionally be improved by using a higher amount of cameras, however, we discovered the improvements not to be significantly. On the contrary, using less than three cameras significantly degrade the quality and is therefore not an option. Also altering the relative camera arrangement can lead to a significant loss of performance.

Using hand-tracking for interaction purposes is undeniably very intuitive and effective. But especially two-handed operations are tiring if the user has to keep her/his arms in free air. Therefore a desk is placed under the working volume in order to let the user rest her/his elbows on it. Furthermore, in this way the natural tremor of the hands is reduced.

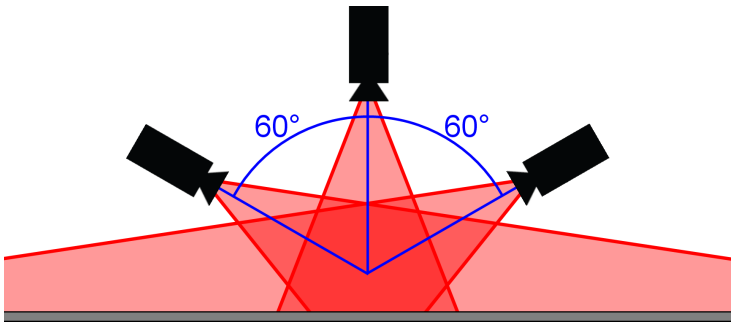


Figure 2.7: Illustration of the confined camera arrangement. The red cones indicate the viewing volumes. Note that the region where all cones overlap each other is defined as the working volume.

2.5.1 Active Infrared Night Vision Cameras

In our high quality setups the user's hands' movements are captured by three video cameras, respectively. The cameras operate in night shot mode and the lenses are equipped with high-pass near-infrared (NIR) filters. The bottom of the working volume is covered with NIR-black material. The volume is illuminated by spotlights in the NIR-spectrum. This setup makes the system almost insensitive to ambient light changes or the absence of light in the visible spectrum.

Figure 2.8: The high quality camera setups of our tracking system. In both setups the cameras are equipped with NIR filters and NIR spotlights illuminate the working volume. Left: consumer camcorders. Right: industrial cameras.

In the first prototype (see Fig. 2.8(Left)) consumer camcorders (Sony DCR-HC94E, $720x576$ pixel at 25 frames per second, unit price: approx. 600 Euros)

are used. Using this setup instant interaction is already possible. However, the low camera frame rate induces a latency of up to 40 milliseconds which is slightly disturbing in some interaction applications (e.g. games). To this end, we built another prototype (see Fig. 2.8(Right)) using industrial cameras (The Imaging Source DMK 21AU04, $640x480$ pixels at 60 frames per second, unit price: approx. 330 Euros). The slightly reduced pixel resolution affect the tracking accuracy only marginally. But with the higher frame rate of this system the latency is hardly perceptible and interaction applications needing fast response (e.g. games) can suitably be controlled. Note that both camera types does not provide thermal vision as they are standard CCD-cameras measuring the reflected amount of radiance per pixel in a spectral range of 700nm - 1000nm and rely on active illumination in the same spectrum.

2.5.2 Low Cost Webcams

In our low cost prototypical setup (see Fig. 2.9) the user's hands' movements are captured by three consumer webcams (Creative Webcam Live Ultra for Notebooks, up to $640x480$ pixel at 30 frames per second, unit price: approx. 20 Euros). This can be seen as a proof of concept. The bottom of the working volume is cov-

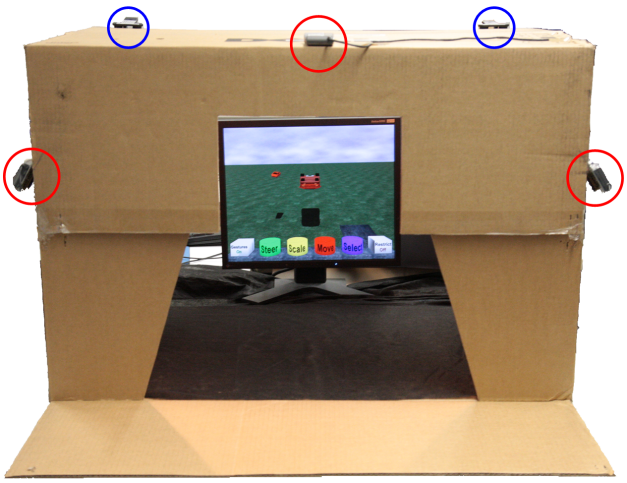


Figure 2.9: The low cost camera setup of our tracking system. The red circles indicate the webcams, the blue ones the spotlights.

ered with black material. As standard color cameras are by default equipped with infrared cut-off filters they can not be run in night shot mode. Therefore, we use a simple color based background subtraction algorithm for segmentation. A pixel is classified to belong to the hand mask if it is significantly brighter than the learned background value. By segmenting only if the pixel is brighter the shadows of the

hands are not falsely segmented as they appear darker than the background. Note that color based background subtraction is sensitive to ambient light changes and enough light is needed for illuminating the user's hands in the working volume. Therefore, the webcams are mounted on a partly enclosed box which excludes most of the ambient light. Additionally, two lights are added to the box which provide a constant illumination.

While webcams provide high resolutions and frame rates at very low prices they are unfortunately not designed for being used at one computer simultaneously. Most manufacturer do not provide information about this functionality. Therefore, we tested several models until we found webcams working together properly. Additionally, each of these webcams has to be connected to a different USB-controller, respectively.

CHAPTER 3

TRACKING ONE HAND

The key idea of our hand-tracking method is to determine for every posture three unique feature points from which the hand pose is deduced. In contrast to previous methods the feature points are not computed in the 2D image domain but in three dimensions. This is done by first computing the visual hull of the segmented images of the different cameras. This way, the needed information for estimation of posture and pose, contained in the images, is combined in one unique representation and complex triangulation procedures are avoided. In addition, 3D-based feature descriptors incorporating e.g. geodesic distance approximations can be used which are more robust than purely image based ones. To the best of our knowledge, such a system was not available before.
Our method is able to robustly track the global position and orientation of a user's hand while distinguishing between 4 predefined postures (see Table 3.1).

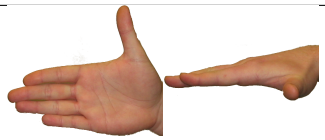
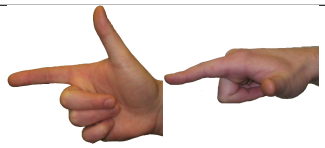
name	**posture**	**tracked DOFs**
palm		pose (6DOF)
picking		pose (6DOF) + fingertip directions
pointingA		pose (6DOF) + fingertip directions
pointingB		position + pointing direction (5DOF)

Table 3.1: Postures identified by our system.

3.1 The Method

For every target posture (see Table 3.1) we can define "most protruding" fingertips as depicted in Table 3.2. These protruding fingertips are of interest to us, because:

- If the current frame can be labeled with the identity of the most protruding fingertips the posture groups 'palm', 'picking'/'pointingA' and 'pointingB' can be recognized.
- If the direction of the corresponding fingers is also known, the ambiguity between the postures 'picking' and 'pointingA' can be resolved.
- If additionally the position of the protruding fingertips and the center of gravity of the hand is known, we can determine the pose (position and orientation) of the hand.

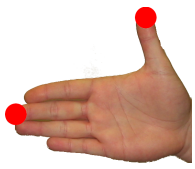
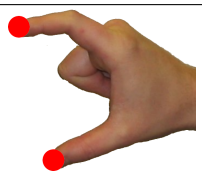
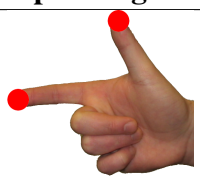
'palm'	'picking'	'pointingA'	'pointingB'
			

Table 3.2: The "most protruding" fingertips of the postures.

We developed an algorithm that is able to extract all the information needed to solve both the posture-recognition and the pose-estimation problem from the camera images in real-time, thus realizing the tracking of the hand. The information is computed from the 3D binary voxel grid of the visual hull, created from the segmented views of the cameras.

The algorithm flow of our hand-tracking system is depicted in Fig. 3.1. First the grabbed images are segmented to obtain the hand/arm masks for the different camera views. Based on the area of the segmented masks we can decide whether the hand is present in the working volume. If the hand is present the masks are combined into a 3D voxel grid (visual hull). Depending on the current state (lost or tracking) we choose between two methods to find the fingertips. Although the computations on the 'lost' branch of the algorithm flow can also be carried out in real-time, the 'tracking' branch (if successful) is much less computationally intensive. Thus, in 'tracking' state we are able to further reduce the delay between the user's action and its interpretation by the application that uses the tracking.

On the 'lost' branch the visual hull computation is followed by finding a small set of potential surface feature points ('lost features' step). Then one or two of

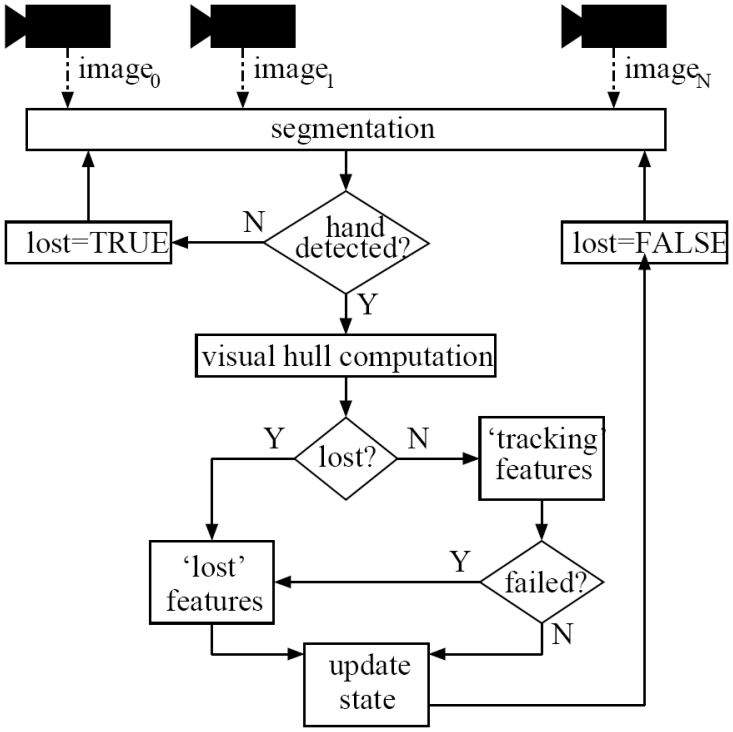


Figure 3.1: Overview of our tracking algorithm.

these points are selected as corresponding fingertips. Which fingertip(s) is(are) detected depends on which posture the user makes, see Table 3.2. The exact fingertips are identified during the 'update state' processing step.

If 'lost features' succeeded, the 'update step' will be carried out. Here, the orientations (pointing directions) of the appropriate fingers are computed from fingertip-local volume properties and also the fingertips are classified. Based on these computed values the current posture is determined and the pose of the hand is computed.

On the 'tracking' branch not the entire set of feature candidates is determined. Based on the hand state (hand position and finger orientations) in the previous frame, only the possible feature points on the new visual hull are computed (assuming a constant velocity model). A simple verification can reveal whether the points found could really belong to the previously detected fingertips. If this is the case, the algorithm proceeds to the same 'update step' as on the 'lost' branch. If the fingertip-verification fails, the 'lost features' step is proceeded.

3.1.1 Computing the Visual Hull

The segmentation of the images is followed by finding the largest connected component in each image. Its bounding box defines the region of interest (ROI) of the image. If the area of hand/arm mask is smaller than a threshold (we used 0.5% of the image area), the given image is not used for further processing (the user's hand is currently not observed by the given camera). If less than 3 images are available further processing, no tracking will be done; the hand of the user is assumed not to be present in the working volume.

The visual hull computation (see Sec. 2.1) is implemented on the GPU as described in [LMS03]. The cells of an equidistant voxel grid are projected to each camera image and depending on the pixel values the voxel is set to filled or empty. This procedure can easily be parallelized on the GPU. The grid has a resolution of $128 \times 128 \times 128$ and is axis-aligned in the world coordinate system. In order

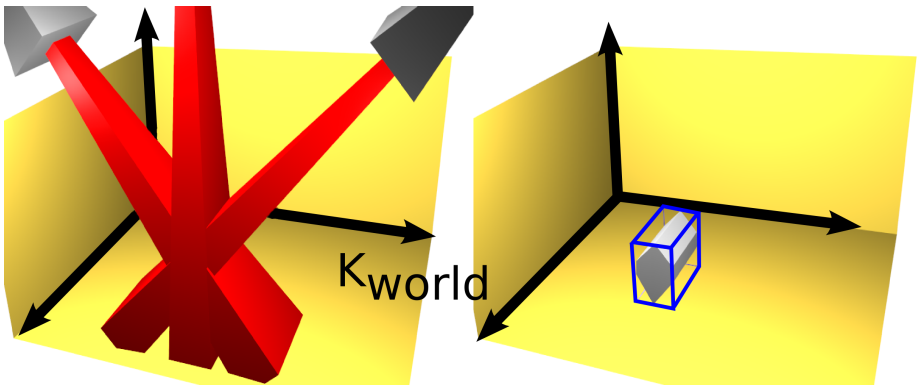


Figure 3.2: Determining the position and extend of the visual hull target voxel grid (K_{world} is the world frame).

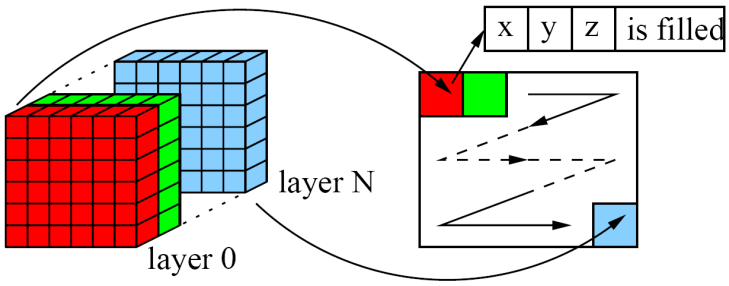


Figure 3.3: Packing of the voxel grid into a 2D texture.

to fully exploit this resolution, the placement of the grid is recomputed in every frame: its extent and position is determined by the axis-aligned bounding box of

the polytope defined by the visual hull of the ROIs of the images (see Fig. 3.2). As the bulk of our tracking is implemented on the GPU and because of hardware restrictions of many GPUs regarding 3D textures, we store the 3D voxel grid as a 2D texture as shown in Fig. 3.3 to be used as input for the subsequent processing steps on the GPU.

3.1.2 Finding Fingertip Candidates

Per definition the fingertips of interest belong to "most protruding" parts of the hand. Therefore, it can be assumed that these fingertip points are located on the boundary of the convex hull of the hand. Unfortunately, computing the convex hull of the visual hull is too complex for real-time applicability. For this reason, only a small subset of visual hull voxels belonging to the boundary of the convex hull is computed.

Candidate fingertips are defined as the visual hull voxels, that touch one of the planes of the bounding discrete orientation polytope (DOP) with 26 suitably oriented planes (see [KHM+98]). We denote the set of these extremal voxels as the 26-DOP points. The 26 normal vectors of the DOP planes are the elements of the set $\{-1,0,1\} \times \{-1,0,1\} \times \{-1,0,1\}$ except the null vector. The signed distance of a point p_i to a plane with the normal vector d_k containing the origin is the scalar product $\langle d_k, p \rangle$. We generate a signed distance texture from the visual hull for half of the normal vector set. Generating the mipmaps of these signed

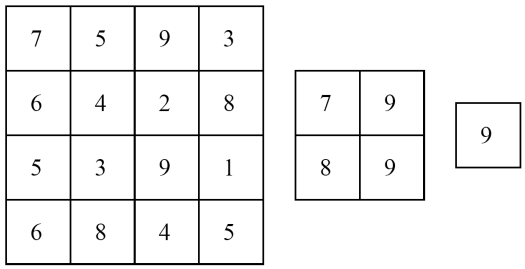


Figure 3.4: Maximum search in a texture via mipmapping.

distance textures using minimum and maximum filtering yields for each d_k the signed distance of the corresponding two 26-DOP planes (see Fig. 3.4). Additionally, the voxel index is transcribed to the mipmap textures with the result that the respective 26-DOP point as well is contained in the highest mipmap level. If several points have the same distance value in the mipmapping process, an arbitrary one is chosen. Moreover, if a point is maximal to multiple d_k, it is taken only once. Note, that the resulting set of fingertip candidate points F_p consists of

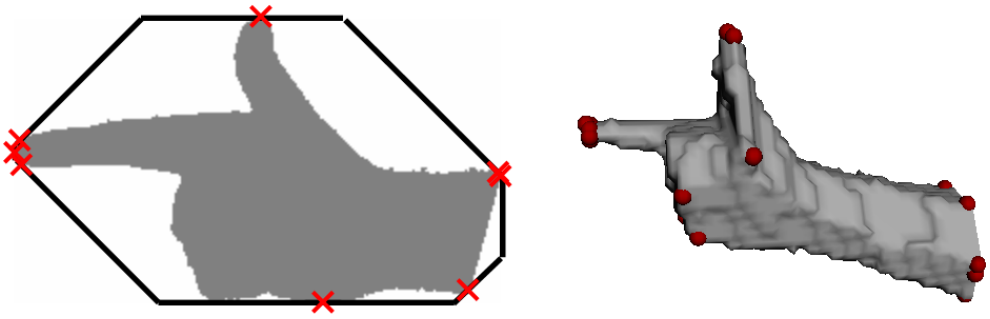


Figure 3.5: Extraction of DOP points. Extremal points are shown in red. Left: 8-DOP points of a silhouette image in 2D. 8-DOP shown in black. Right: 26-DOP points of the visual hull in 3D. Note that several fingertip candidates are located on the index-finger.

at most 26 points (see Fig. 3.5) all being located on the boundary of the convex hull of the visual hull.

3.1.3 Identifying Fingertips

For identifying the fingertips first the candidate points are ranked, then duplicates and false candidates are filtered out and finally the fingertips and hand posture are classified.

Ranking Fingertip Candidates

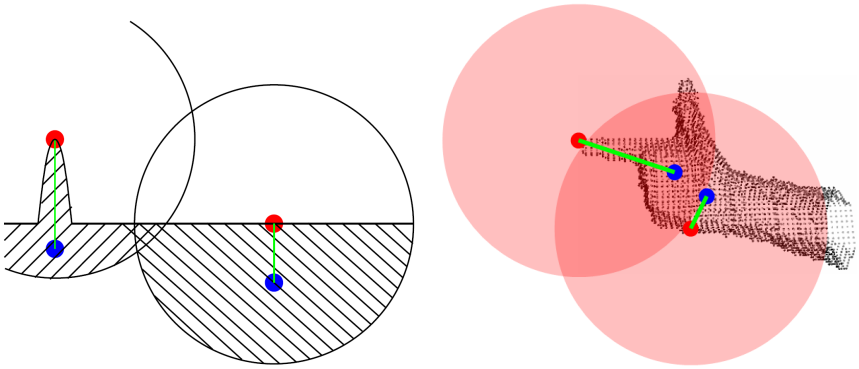
We can think of the fingertips as endpoints of protruding parts of the voxel grid. In order to be able to rank the candidate points $p_i \in F_p$, we have to define a measure of protrusion D_i. Our choice for this measure is the distance of p_i to its local center of mass m_i measured in a ball $B(p_i, \sigma)$ of a certain radius σ. Note that the radius is always greater than the length of the protrusion. We expect D_i to be maximal for endpoints of protruding forms (see Fig. 3.6). Formally, D_i is defined as

$$D_i = \|p_i - m_i\| \tag{3.1}$$

with

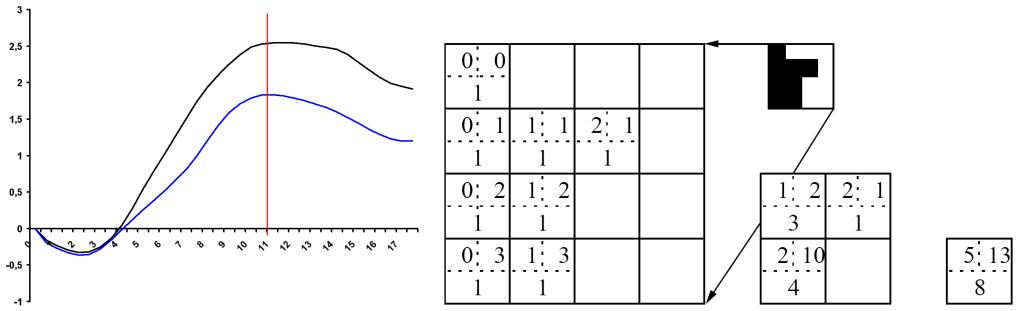
$$m_i = \frac{\sum_{v_j \in B(p_i,\sigma)} (v_j \psi(v_j))}{\sum_{v_j \in B(p_i,\sigma)} \psi(v_j)}, \tag{3.2}$$

where v_j denotes the position of a grid voxel and $\psi(v_j)$ is defined to be $\psi(v_j) = 1$ if v_j belongs to the visual hull (voxel is filled) and $\psi(v_j) = 0$ otherwise (voxel is not filled).



(a) 2D (b) 3D

Figure 3.6: Measuring "protrudingness" D_i by taking the distance of a candidate point (red) to the local center of mass (blue).



(a) Distinctiveness of Protrudingness (b) Local Center of Mass Computation

Figure 3.7: a) Distinctiveness (y-axis) of our "protrudingness" measure for different radii (x-axis). The black curve corresponds to the index-finger, the blue one to the thumb. The vertical red line marks the chosen radius. b) Calculating the local center of mass using mipmapping. The base level encodes the coordinates of each pixel/voxel in the RG channels, while the alpha channel holds the filled/non-filled info. After constructing the mipmap using additive filtering, the final mipmap level contains the number of mask pixels in the alpha channel by which the coordinates in the RG channels must be divided.

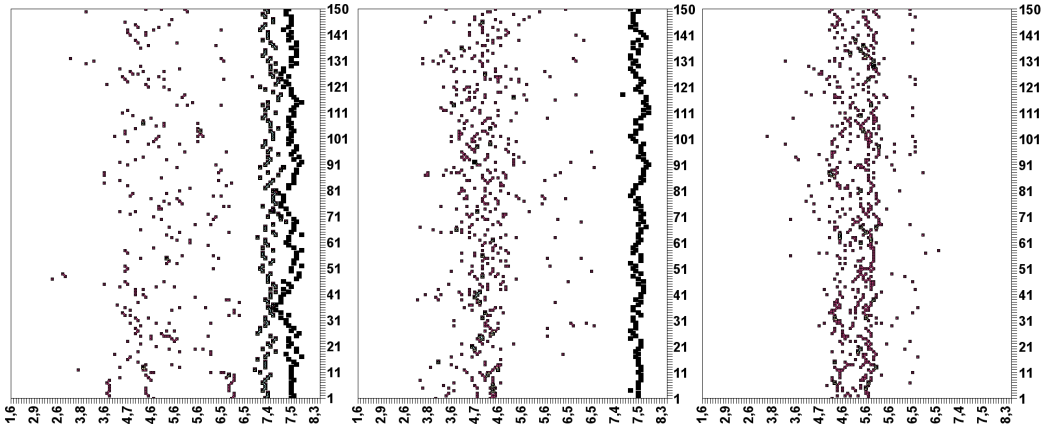


Figure 3.8: Histograms of the "protrudingness" D (x-axis) for 150 frames (y-axis) in 'pointingA' posture (Left), 'pointingB' posture (Middle) and a fist posture (Right).

To determine σ, we examined the distinctiveness of D_i for several σ values ranging from 0 to 18 centimeters. Therefore we computed the difference between the D_i values of the index-fingertip and another candidate point on the arm and the same difference for the thumb-tip for a representative hand-sequence. As our goal is separating fingertip points from points being located on the arm, the optimal radius is identified when both differences are approximately maximal (see Fig. 3.7(a)). Even though we performed this measurement with only one representative hand, our chosen σ is well-suited for all adult hands and still practicable for most children's hands.

We compute local centers of mass on the GPU with the help of mipmapping. The basic idea for this algorithm is explained in 2D in Fig. 3.7(b).

To analyze our "protrudingness" measure, we computed histograms of 150 frames of a hand moving in the 'pointingA' posture, in the 'pointingB' posture and in a fist posture, which are shown in Fig. 3.8. There, the entries of the one fingertip for the 'pointingB' posture and the entries of the two fingertips for the 'pointingA' posture are emphasized. Obviously, the fingertip points have significantly greater "protrudingness" values, what demonstrates the capability to separate fingertips from other points.

Filtering out Duplicates and false Candidates

Since more than one candidate can correspond to the same finger (see Fig. 3.5), we have to filter out duplicates. This is done by removing each candidate for which another candidate with the following properties exist:

1. It has a major D_i-value.

2. It is closer than a certain threshold (we used 5 centimeter).

3. The line segment connecting the two candidates does not leave the visual hull for more than one centimeter. This avoids mistaking candidates on two not touching fingers for being duplicates.

In Fig. 3.9 an exemplary result of this filtering is shown.

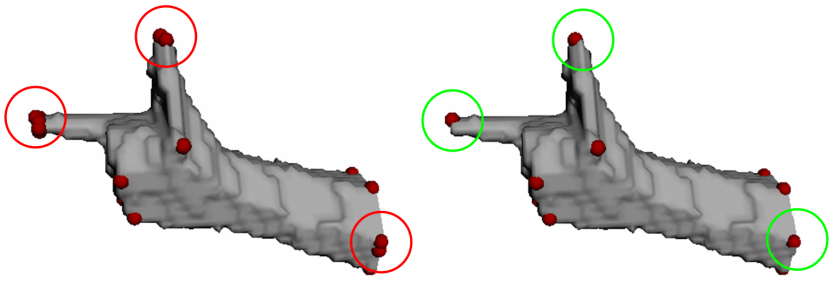


Figure 3.9: Result of filtering out duplicates. Left: Beforehand. Right: Afterwards. Several points located inside the green circles were filtered out.

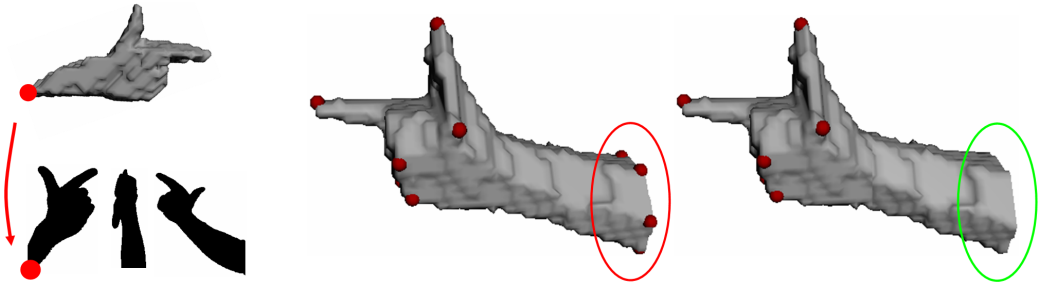
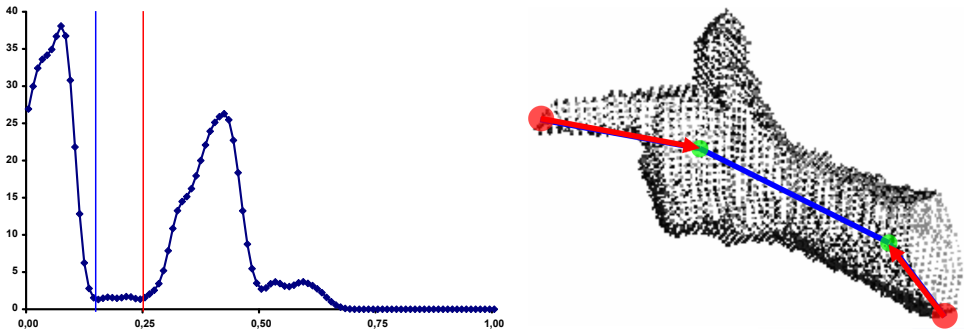


Figure 3.10: Left: projected fingertip candidate point (red, top) is close to image border (red, bottom). Middle: example with all fingertip candidates. Right: example with filtered fingertip candidates.

The two elements of F_p, which have the highest D_i-values D_1 and D_2 are our choice for fingertips. In practice we found that the segmentation can cut off the arm sharply producing false fingertip candidates. Therefore we ignore points whose projected image positions are close to the image border (see Fig. 3.10) when determining D_1 and D_2.

Classifying the Fingertips and the Posture

If the hand forms the 'pointingB' posture, the ratio D_2/D_1 should be smaller than in the case of the other postures, where the thumb is also present. We verified this assumption by plotting a histogram of our measure for the 'pointingB' posture $\rho_{pointingB} = 1 - \frac{D_2}{D_1}$ for various tracking sequences (see Fig. 3.11(a)). This also helped us to obtain suitable lower and upper hysteresis thresholds to differentiate between the 'pointingB' posture and the other postures.



(a) Histogram of $\rho_{pointingB}$ (b) Geodesic Distance Approximation

Figure 3.11: a) Histogram of the ratio $\rho_{pointingB}$ (x-axis) for various tracking sequences. The blue/red vertical line illustrates the lower/upper hysteresis threshold. b) Approximation of the geodesic distance. In this example the path of the geodesic distance approximation between the front fingertip and a point on the arm is shown. The red arrows indicate the distances $\|p_i - m_i\|$ and $\|m_j - p_j\|$, respectively, and the blue line indicates the distance between the mass points $\|m_i - m_j\|$.

If two fingertips p_1 and p_2 are present we have to decide which one is the thumb. Identifying the thumb is based on the observation that the maximal geodesic distance between the thumb fingertip and all other candidates (including the false candidates on the arm) is smaller than the similarly computed maximal geodesic distance for the front fingertip. Since calculating the exact geodesic distance is prohibitive for real-time use, we have to approximate it. Let m_i and m_j denote the already computed local centers of mass for candidate points p_i and p_j, respectively, we approximate the geodesic distance between p_i and p_j with

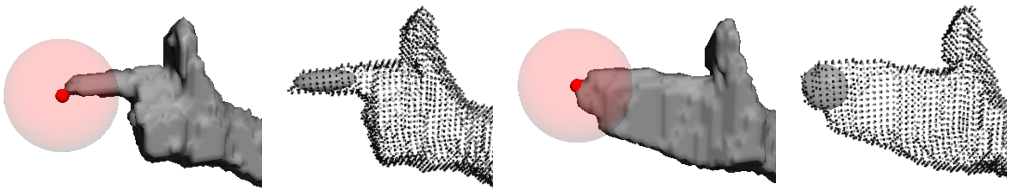
$$\begin{aligned} d_{geo}(p_i, p_j) &\approx \|p_i - m_i\| + \|m_i - m_j\| + \|m_j - p_j\| \\ &= D_i + \|m_i - m_j\| + D_j. \end{aligned} \tag{3.3}$$

This is illustrated in Fig. 3.11(b). Now, the fingertip for which the maximal approximated geodesic distance to all points in F_p is smaller is identified as the

thumb-tip p_{thumb}, which can formally be expressed as follows:

$$p_{thumb} = \begin{cases} p_1, & \text{if } \max_{p_j \in F_p}(d_{geo}(p_1, p_j)) < \max_{p_j \in F_p}(d_{geo}(p_2, p_j)) \\ p_2, & \text{otherwise} \end{cases} \tag{3.4}$$

Next we have to decide whether the non-thumb finger is the middle or the index-finger in order to distinguish posture 'palm' and 'pointingA'. We achieve this by computing the covariance matrix locally around the fingertip with a GPU algorithm similar to the one we used for computing the local center of mass. The

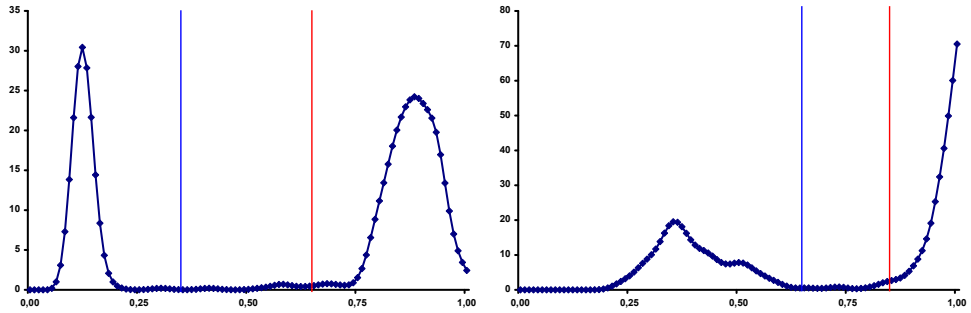


(a) (b) (c) (d)

Figure 3.12: a) and c) Local balls for PCA computation. b) and d) Ellipsoids corresponding to the eigenvalues and eigenvectors. Note that for posture 'pointingA' the ellipsoid is cigar-shaped while it is disc-shaped for posture 'palm'. This shape variation can be measured by the ratio of the eigenvalues of the covariance matrix.

ratio $\rho_{palm} = \frac{\lambda_2}{\lambda_1}$ between the highest eigenvalue λ_1 and second highest eigenvalue λ_2 of the covariance matrix allows us to establish the identity of the non-thumb finger (see Fig. 3.12). In Fig. 3.13(a) a histogram of ρ_{palm} and the used hysteresis thresholds are depicted that illustrates the capability of using ρ_{palm} to distinguish these two postures. Note that a high ratio indicates the posture 'palm'.

Finally, if the non-thumb finger turned out to be the index-finger, we have to distinguish between the 'pointingA' and the 'picking' posture. Commonly, if people pick something with the index-finger and the thumb, the last phalanx of the index-finger and the thumb is approximately parallel. Additionally, the line connecting the two fingertips is approximately orthogonal to the direction of the last phalanxes of the two fingers. This is illustrated in Fig. 3.14(Left) and used to determine the posture as described in the following. First, we approximate the direction of the last phalanxes of the two fingers. Let m_{finger} denote the local center of mass computed with a small radius (2.5 cm) around the according fingertip, the difference $d_{finger} = p_{finger} - m_{finger}$ is assumed to be the direction of the last finger phalanx. Then, we use the this way computed directions $d_{index-finger}$ and d_{thumb} as well as the direction of the line connecting the fingertips $d_{line} = p_{index-finger} - p_{thumb}$ to define a measure for picking $\rho_{picking}$ as



(a) Histogram of ρ_{palm} (b) Histogram of $\rho_{picking}$

Figure 3.13: Histograms for various tracking sequences of the values that are used to distinguish the 'pointingA'/'palm' postures (a) and the 'pointingA'/'picking' postures (b). The blue/red vertical lines illustrate the lower/upper hysteresis thresholds, respectively.

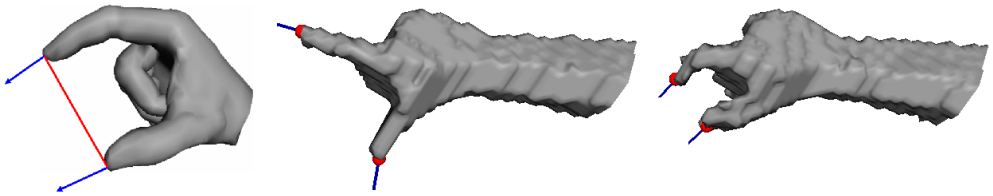


Figure 3.14: Left: Illustration of a conventional grabbing posture. In blue the directions of the last phalanxes of the fingers $d_{index-finger}$ and d_{thumb} are shown, and in red the line connecting the fingertips is depicted. Middle: Directions in 'pointingA' posture. Right: Directions in 'picking' posture.

follows:

$$\begin{aligned} \rho_{picking} &= 1 - \frac{\alpha_{index-finger} + \alpha_{thumb}}{\pi}, \quad \text{with} \\ \alpha_{index-finger} &= \max(0, \angle(d_{index-finger}, -d_{line}) - \frac{\pi}{2}), \\ \alpha_{thumb} &= \max(0, \angle(d_{thumb}, d_{line}) - \frac{\pi}{2}), \end{aligned} \tag{3.5}$$

where $\angle(\cdot,\cdot)$ gives the angle (in radians) between two directions. Fig. 3.13(b) shows a histogram of $\rho_{picking}$ and the according hysteresis thresholds we used for distinguishing between the 'pointingA' posture (low $\rho_{picking}$ value) and the 'picking' posture (high $\rho_{picking}$ value).

As we are using hysteresis thresholding to control posture transitions falsely identified posture changes are minimized. All lower and higher hysteresis thresholds were identified analyzing recorded hand movement sequences. Because of

the chosen criteria, the optimal thresholds depend only loosely on the individual. Therefore, using the same constant thresholds for every user is sufficient and leads to robust results.

3.1.4 Exploiting Coherence

In order to reduce the delay introduced by the tracking, we try to reuse the information from the previous frame provided they were valid. This way we try to avoid the full feature detection procedure described in the previous section. Assuming a constant velocity model, as long as the current posture does not change, the current fingertip positions are located in the vicinity of the previous fingertip positions. Also, the finger directions should not change significantly.

Therefore, in case of postures 'pointingA', 'picking' or 'palm', we concentrate only on the computation of the two current fingertips, instead of computing a whole set of fingertip candidate points. For each previous fingertip point p_i with previous finger direction n_i we search in the region of a local ball $B(p_i, \sigma)$ with $\sigma = 4cm$ for the voxel v_j for that $\langle v_j, n_i \rangle$ attains its maximum. Therefore we set

$$p_i' = \textbf{argmax}_{v_j \in \{v_k \in B(p_i,\sigma) | v_k \text{ is filled}\}} \big(\langle v_j, n_i \rangle \big). \tag{3.6}$$

This procedure is illustrated in Fig. 3.15.

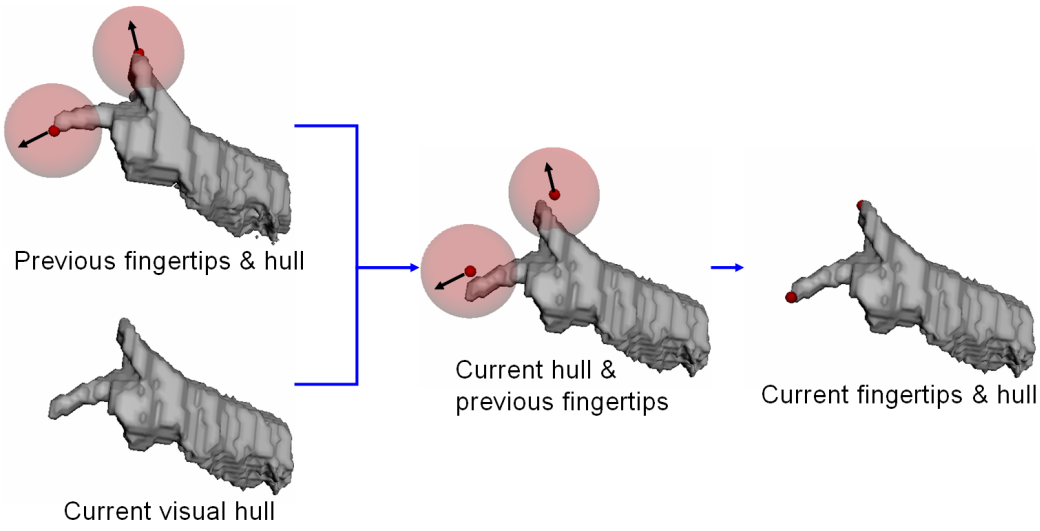


Figure 3.15: Illustration of how coherence can be exploited.

If the set $\{v_k \in B(p_i, \sigma) | v_k \text{ is filled}\}$ is empty or p_i' is located on the boundary of the ball $B(p_i, \sigma)$, we assume that the new position p_i' is not valid any more. If the new thumb fingertip as well as the new non-thumb fingertip are valid, we can

skip the 26-DOP computation (see Sec. 3.1.2) and perform the steps described in Sec. 3.1.3 only partially; just the parts of checking if the user has changed the posture are needed.

Unfortunately, for the 'pointingB' posture the simple tracking does not work, since then there is only one current fingertip and we always have to check if the user has changed the posture.

In case that any of the fingertips became invalid, all steps of the pipeline, starting with the extraction of fingertip candidate points are performed.

Note, that the computation of p'_i can be performed on the GPU analogously to the computation of the 26-DOP points. Fortunately, in this case a significantly smaller part of the visual hull has to be considered. Since the computation of all fingertip candidate points is not necessary and the "protrudingness" measure has to be evaluated only for the two fingertip points, this results in a great speed up as shown in Table 3.3.

3.1.5 Estimating the Pose

Having estimated the hand posture, we still have to deduce position and orientation of the hand, for which a third point is needed. Note that for stable tracking of the pose it is important that this point remains at the same relative position over time. Therefore, we decided to use the center of mass of the hand as a third point. The center of mass can be computed as the local mass point of one of the fingertips when an adequate σ' chosen.

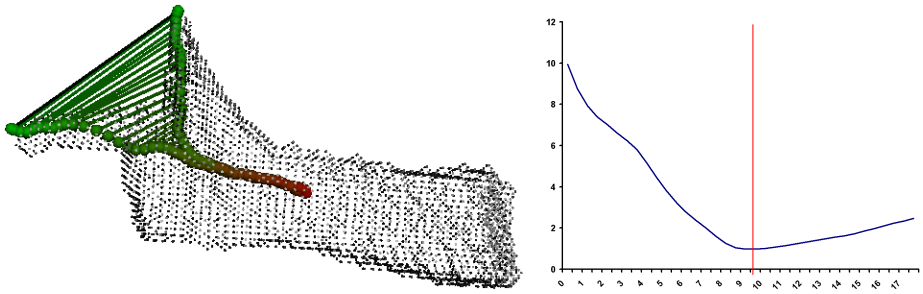
To derive σ', we ran the following procedure: we computed for the thumb and front fingertip the local center of mass with different σ. At the same time, the distance $\|m^{\sigma}_{front} - m^{\sigma}_{thumb}\|$ of the local mass points is measured. As depicted in Fig. 3.16, a clear minimum exists, what we think to be the best choice for σ'. Following this, our hand center point p_{hand} is defined as $p_{hand} = m^{\sigma'}_{front}$.

Because of the natural tremor of a human hand and some jitter in the 2D images the feature point positions (fingertips and hand center) tremble slightly. In order to compensate for this, for each feature point we use a Gaussian kernel with $\sigma = 1cm$ located at the fingertip's current position to compute a spatially weighted average of its last five positions (including the current one). While this kind of smoothing improves the subjective impression significantly, the induced latency is marginal.

Now, we use the normalized vectors $o_1 = \frac{p_{thumb}-p_{hand}}{\|p_{thumb}-p_{hand}\|}$, $o_2 = \frac{p_{front}-p_{hand}}{\|p_{front}-p_{hand}\|}$ and $o_3 = \frac{o_1 \times o_2}{\|o_1 \times o_2\|}$ to derive the global hand orientation matrix O as follows:

$$O = \left[(o_2 \times o_3)\,(o_2)\,(o_3)\right]. \tag{3.7}$$

In the case of posture 'pointingB' we use p_{hand} as the position and $p_{front} - p_{hand}$ as the direction to determine 5 DOFs.



(a) (b)

Figure 3.16: Determination of σ' used for the hand center computation. (a) Example of local mass points with different σ. Mass points computed with same σ are connected by a line and have the same color. (b) Distance (y-axis) of local mass points of front and thumb fingertip with different radii (x-axis). The vertical red line marks the chosen radius.

3.1.6 Timings

The run times of our system for one frame are depicted in Table 3.3. Thereby, all cameras are connected to one computer (see Sec. 2.5), where the images are segmented on the CPU. Thereby, OpenMP (Open Multi-Processing) is used to parallelize the computations. The resulting segmented regions of interest are loaded to the GPU, where most of the other computations are performed.

Table 3.3: Timings (in ms)

Procedure	**Lost**	**Tracking**
Image segmentation (CPU)	6	6
Loading 3 binary images to GPU	1-2	1-2
Visual hull computation (GPU)	1-2	1-2
Fingertip candidates (GPU)	14	1
Fingertip selection (GPU)	2	1
Posture and pose estimation (GPU and CPU)	2-4	2-4
Sum	25-29	12-16
Mean (averaging several sequences)	27	13

3.2 Limitations

Unfortunately, some limitations remain by using this interaction setting. In particular the need for a controlled environment in order to guarantee good segmentation

results restricts the feasibility in some application fields. Additional techniques for segmenting the hand would be needed to reduce these requirements, but this was not the focus of our work.

Thinking about practical use in front of large displays, it would be desirable to have an adequately large working volume. To solve this without affecting the tracking accuracy, either more cameras or cameras with a higher resolution are needed.

In our method only four different postures are recognized, which at a first glance seems to be insufficient for designing a complex application. But in practice, we observed the intuitiveness to decrease heavily if the number of different postures used in the same application increases. Moreover, permitting more postures at the same time will always increase the false positive rate in the posture recognition process. Nevertheless, if a more expressive and diverse posture alphabet is provided, different interaction interfaces could use different (small) subsets of postures in order to get the most intuitive mapping between real and virtual actions. In our method, we use heuristics based on the fingertip points to distinguish between different postures, what turned out to be stable even if hands of different size and shape use the system in the same setting. But unfortunately, this leads to a high complexity for extending the set of permitted postures. Because of this, if an application needs many additional postures, other approaches for posture recognition as for example machine learning based techniques could be a solution.

CHAPTER 4

TRACKING TWO HANDS

In this chapter the one-hand-tracking method from the previous chapter is extended for the simultaneous tracking of the user's left as well as the user's right hand.

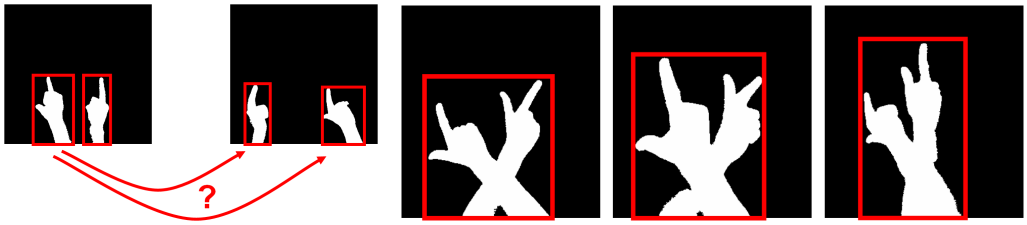
4.1 Overview

In order to simultaneously track both hands two instances of the one-hand-tracking method are running in parallel, one for the right and one for the left user hand. Each instance gets three segmented image sections corresponding to one hand as the input with which the procedure computes the according hand pose/posture as the output. This reduces the problem of simultaneously tracking both human hands to the following parts:

Distribution of the Regions of Interest: If two hands are currently located inside the working volume, each camera image can contain two separable connected components each corresponding to one hand, respectively. Each component defines one regions of interest (ROI) (see Sec. 3.1.1). In order to consistently assign these ROIs to the two one-hand-tracking procedures, correspondences between the ROIs of the different camera images have to be determined as illustrated in Fig. 4.1(a).

Discovery of Left/Right Hand: It has to be determined which tracked hand parameters (pose and posture) belongs to the left and which belongs to the right user's hand. This can be done after the individual tracking of each hand is completed.

Prohibition of Unstable Tracking: If no camera image can be split into two ROIs either only one hand is located inside the working volume or the two hands partly occlude each other in each camera image (see Fig. 4.1(b)). In the latter case, stable tracking of even one hand can not be guaranteed, therefore this case should at least be detected for prohibiting unstable tracking.



(a) (b)

Figure 4.1: (a) Correspondences between the regions of interest are needed. (b) The two hands occlude each other in each camera image.

In the following three subsections our solutions to these problems will be described. An illustration of how the solutions work can be seen in the accompanying video.

4.2 Distribution of the Regions of Interest

In the two-handed case, in a segmented camera image either both hand regions are contained in the largest connected component (LCC) (one of the hands occludes the other) or the LCC contains one hand and the second LCC contains the other hand (no hand is occluded by the other). Therefore, we compute both LCCs as the regions of interest (ROIs) of the images, but reject those LCCs for which the areas of the hand/arm masks are smaller than a certain threshold (we used 0.1% of the image area). This results in three possible cases:

1. One or more images contain no ROI. The hand of the user is assumed not to be present in the working volume. No further processing will be done.

2. All images contain exactly one ROI. Only one hand of the user is assumed to be present in the working volume. The hand is tracked by the one-hand-tracking method.

3. At least one image contains two ROIs. Both hands of the user are assumed to be present in the working volume.

In the third case, we have to decide which image ROIs belong to which hand. Therefore, we compute the polytopes defined by the visual hulls of the ROIs of the images (see Fig. 4.2) and select two polytopes as follows: first, we assign to each polytope all ROIs, that include its projected polytope center. Note that one ROI is assigned to all polytopes, if the image contains only one ROI. Then we identify all pairs of polytopes, whose assigned ROIs comprise the whole set of

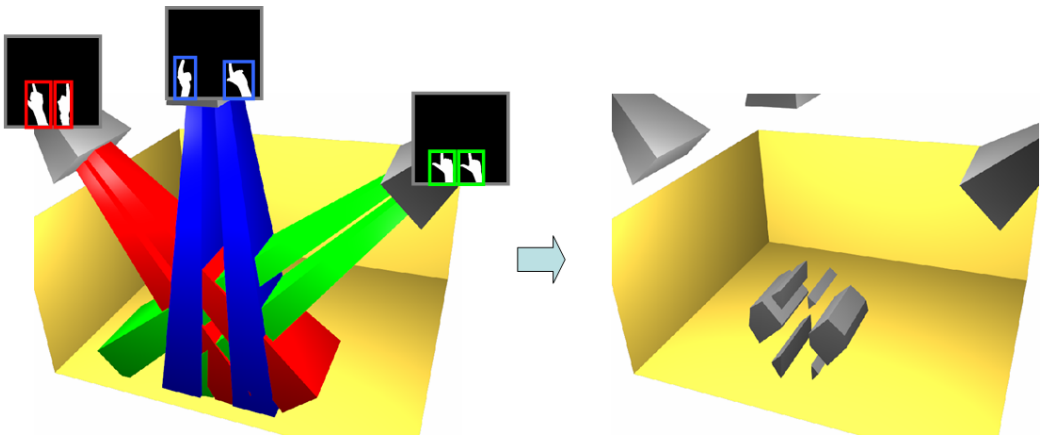


Figure 4.2: Illustration of the construction of the polytopes defined by the visual hulls of the ROIs of the images. The image ROIs define visual cones in the 3D space (Left) and the cones' intersections define the polytopes (Right).

ROIs. After that, we select the pair with the largest sum of the polytope volumes and use each of the two assigned sets of ROIs to be further processed by one of the one-hand-tracking instances to compute the hand pose and posture.

In order to let the individual tracking of each hand exploit frame to frame coherence as explained in Sec. 3.1.4, left and right hand have to be tracked consistently across frames. Thus, the sets of ROIs have to be associated correctly with the two instances of the one-handed pose and posture recognition procedure. Assuming both hands to be successfully tracked in the previous frame, we minimize the sums of the squared distances of the current polytope centers c'_1 and c'_2 to the previous polytope centers c_1 and c_2. That means, if $(\|c_1 - c'_2\|^2 + \|c_2 - c'_1\|^2)$ is smaller than $(\|c_1 - c'_1\|^2 + \|c_2 - c'_2\|^2)$, we exchange the two ROI-sets; otherwise they are already associated correctly. In the case that currently only one hand is present or previously only one hand was present in the working volume, the minimization and assignment is done only for the existing polytope centers/ROI-sets.

4.3 Discovery of Left/Right Hand

Having successfully tracked both hands we furthermore have to determine which hand is the left and which is the right one. In the case that previously two hands were tracked successfully, we simply carry this information over to the current situation. In the other case (no coherence to the previous frame can be used), in order not to reduce speed, we decided to use a simple test that works even if the left hand is located further right than the right hand. This test is based on the

observation, that the user's left elbow joint is always located further left than the right elbow joint (except for unnaturally crossed arms). We exploit this by taking into account the computed hand centers c_{H_1} and c_{H_2} and the respective mean m_{H_1} or m_{H_2} of the feature candidate points that were located on the arm where it was truncated by the segmentation (see Fig. 4.3 (Top)). Note that such points were already computed and identified in the one-hand-tracking procedures (see Sec. 3.1.3). Furthermore, we compute the points p_{H_1} and p_{H_2} determined by

$$p_{H_i} := c_{H_i} + 35\frac{m_{H_i} - c_{H_i}}{\|m_{H_i} - c_{H_i}\|} \tag{4.1}$$

and use these points to be the approximate positions of the elbow joints (35 centimeters is the approximate distance between a human hand and elbow joint), see Fig. 4.3 (Bottom). Assuming the left elbow joint to be located further left than the

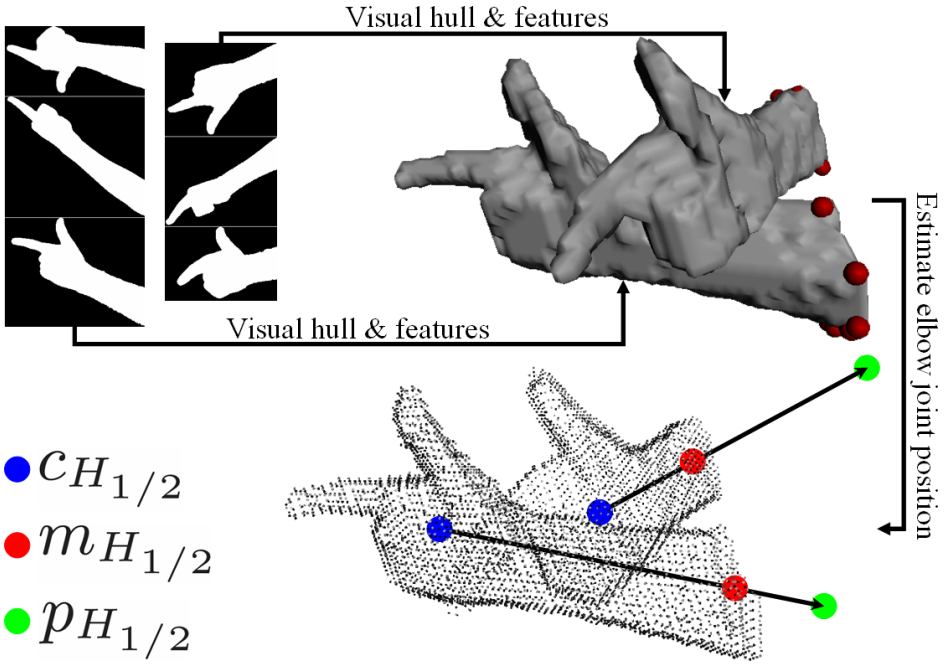


Figure 4.3: Computation of the approximate elbow joint positions.

right one (and vice versa), we simply assign the hands to be left or right accordingly.

4.4 Prohibition of Unstable Tracking

The case that all camera images contain only one ROI can occur even if two hands are currently present in the working volume. This happens when the hand

masks can not be separated in any image (see Fig. 4.1(b)). Consequently we are not able to track both hands and even tracking of only one hand is not robust any more. To prohibit unstable tracking we detect whether one or two hands are currently located in the working volume and in the case of two we discard the tracked pose/posture while indicating the user to increase the distance between her/his hands. To this end we again use the computed feature candidate points that are located on the visual hull where it was cut off by the segmentation (see Fig. 4.3 (Top)). These points are clustered into disjoint sets of points such that the connecting line between any two points in a cluster does not leave the visual hull. This way, all points of a cluster belong to the same arm (see Fig. 4.4). If two

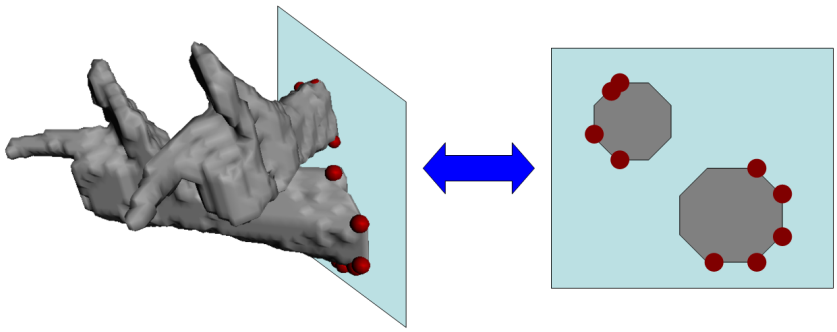


Figure 4.4: The points on the truncated arms belong to different components.

or more sets exist, we assume two hands to be currently present in the working volume.

4.5 Algorithm Flow

The resulting algorithm flow for our simultaneous tracking of two hands is illustrated in Fig. 4.5. After the image segmentation the regions of interest are distributed (red boxes) as described in Sec. 4.2. Then, our one-hand-tracking method is applied for determining the pose/posture either of one or of both hands. If both hands were tracked it is determined which is the left and which is the right hand (blue boxes) according to Sec. 4.3. Otherwise, if our one-hand-tracking method was applied only once, it is tested whether one or two hands were located in the working volume (green box) to prohibit unstable tracking (see Sec. 4.4). Depending on the results of the respective parts either the tracking state is updated or set to lost.

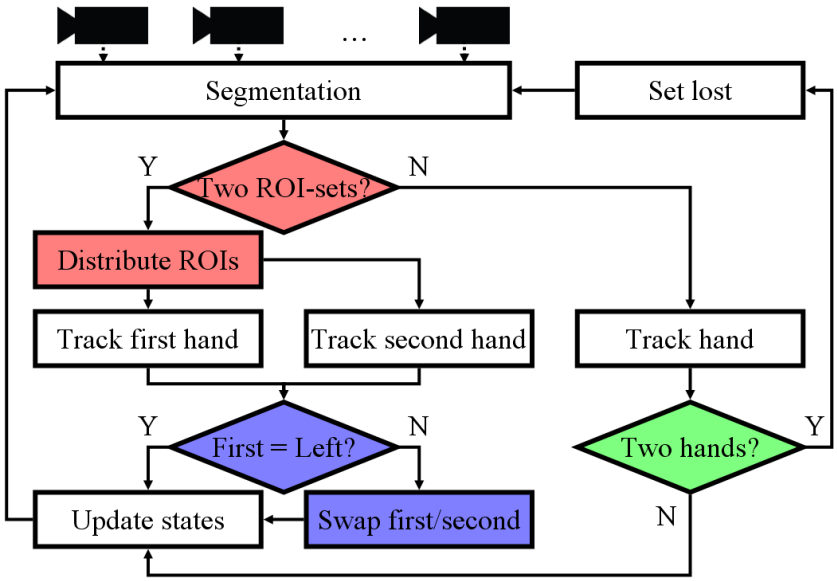


Figure 4.5: Algorithm flow of our two-hand-tracking method. The colored boxes correspond to different sections, respectively.

4.6 Timings

Because the computation of the pose and posture can be performed in parallel on two different computers, the run time of this part does not change. The additionally needed computations require less than one millisecond, so that the overall run time remains almost equal to Sec. 3.1.6.

4.7 Limitations

A current limitation of our approach emerges in the case that two hands are inside the working volume but no camera can separate the two hand masks. Although we can detect this case to prevent unstable tracking it would be desirable to allow such cases in the future. A related problem consists in the decreasing tracking stability, if the number of camera images, which can not separate the two hand masks, increases.

CHAPTER 5

RELATED AND CONCURRENT WORK

First, we want to enlighten the differences between tracking the human body or the human hand. Of course there is a close relationship between hand and body-tracking and many hand-tracking approaches are based on methods used for body-tracking. On the contrary, many differences prohibit the straightforward application of body-tracking methods for hand-tracking. First, the estimation of the human body pose can profit from the hierarchical bone structure (e.g. head, torso, limbs, etc.). The hierarchical structure of the hand consists only of palm and fingers. Second, while the bare human hands lack a distinctive texture, the clothing on human body can support the segmentation due to color and texture features. Third, the concave hand shape and the close proximity of the limbs lead to severe occlusions. Because of the lack of texture the recognition of such occlusions is further complicated. For these reasons, we refrain from reviewing human body-tracking algorithms and refer the interested reader to survey papers as for example [AC99, Gav99, MG01, WHT03, WS03, Hec06, Pop07].
Erol et al. [EBN+07] stated the following major difficulties of designing a suitable hand pose estimation system:

High-dimensional problem: The bone structure of the human hand enables articulations with more than 20 DOFs. Moreover, the global hand position and orientation is arbitrary.

Self-occlusions: Due to the flexibility of hand articulations a 2D image of the hand mostly includes many self-occlusions.

Processing speed: Depending on the application even a latency of more than 30 milliseconds can be too high for direct interaction. However, one or more video streams produce a huge amount of data to be processed in real-time, which prohibits the exploitation of complex algorithms.

Uncontrolled environments: Segmenting the hand from the background in a camera image is still a challenging issue in computer vision, even if some restrictions on e.g. lighting are given.

Rapid hand motion: The hand can be moved with a very high velocity, which makes pure coherence based tracking nearly impossible. Therefore, an additional slower hand pose detection procedure is often used when the faster tracking algorithm lost the hand.

Unfortunately, in the hand-tracking literature still no approach is known that addresses all these issues. All methods make some restrictions with respect to one or more of the above mentioned problems. Depending on the restrictions a hand-tracking system can only be used for specific applications. For example, a system not being capable of tracking in real-time can only be used for motion capturing and not for direct interaction. Real-time tracking means having a per frame processing latency lower than the time between two frames. Otherwise the tracking can only be processed off-line. In the following discussion of hand-tracking approaches we will particularly refer to the respective restrictions that are made.

In the following, several hand pose estimation approaches are shortly outlined according to the taxonomy of Erol et al. [EBN+07]. The main two categories are full DOF (degrees of freedom) and partial pose estimation. We first outline the typical segmentation approaches used for hand-tracking, then highlight a few representative approaches belonging to full DOF tracking and last but not least discuss partial pose estimation approaches most relevant to our work.

However, the large amount of literature in the field of visual hand-tracking makes it practically impossible to give a full review of the previously reported methods here; elaborate analysis can be found in survey papers [WH01, EBN+05, EBN+07, Hec06].

5.1 Hand Localization/Segmentation

To localize the image regions that belong to the hand one of the most common approaches is using skin color segmentation (e.g. [MLN05, OZR02, SSK01, MZ07]), where the segmentation is achieved by thresholding on the image color based on a learned color histogram. For enabling such an approach, the background must not contain objects having a similar color as the hand itself and a good ambient illumination is needed. Moreover, skin color segmentation can generally not be applied for additionally segmenting the user arm, because of possible clothing. Beside color thresholding the segmentation can additionally be improved by combining it with e.g. edge extraction as done in [MZ07].

Another approach is using background subtraction as done in [SK98, ML04, vHB01, LB04], where a before learned background image is subtracted from the current image to obtain a difference image. Then a simple thresholding on the difference image delivers the foreground image regions. Thereby, the background

has to be static and must not contain objects too similar to the foreground (hand and arm). Moreover, a good ambient illumination is needed. Using background subtraction the problem of arm segmentation as mentioned in the previous paragraph can be solved. By using cameras equipped with infrared band pass filters and infrared illumination (e.g. [KSK01]) the illumination can be shifted to an invisible spectrum, which is of great advantage in applications needing a dark environment (e.g. if a projector is used for visualization).

Also a thermal vision infrared camera can be used to solve the segmentation problem (e.g. [OSK02]). This way, the hand can be distinguished from the background by thresholding on the temperature. However, such cameras are still very expensive and heat sources in the background can distort the result significantly.

The use of depth sensing cameras (e.g. [OBL+05, HMF08]) or stereo vision (e.g. [CH96, GBCB00]) is a good solution to overcome most of the restrictions on the background. The hand/arm can simply be segmented based on the distance from the camera(s). However, the currently available stereo vision systems suffer from noisy images and/or lack of performance. Note that the latency induced by stereo vision adds to the overall latency. Active depth cameras can be an alternative, however, current available models suffer from low image resolutions (i.e. less than 200x200) or very high costs (more than 6000 Euros per piece). The most prominent depth cameras are Time-of-Flight (ToF) cameras, which are able to directly capture depth information by measuring the time that light is traveling from an emitter to the camera sensor. Although their resolution is still very low, we argue that they will be the common choice for enabling hand segmentation and simplifying hand-tracking in the near future.

Other approaches for hand localization are using parametrized deformable hand or finger templates [LTCK03, MI00, CH96] or object detection methods based on trained data [VJ01, KT04a, OB04]. Although the results show significant improvements on segmentation, both kinds need a considerably higher amount of computational effort, thus reducing the processing speed.

5.2 Full DOF Estimation

Approaches being able to recover the full global pose as well as the joint angles of the hand can be divided into model-based tracking and single frame pose estimation approaches. As both kinds have other objectives than our approach, we will not go into detail here.

5.2.1 Coherence-Based Tracking

Methods able to track full-DOF tend to be model-based tracking approaches (e.g. [Ste04, Che04, OH99, dLGP06, BKMSG04, dLGPF08]). As they usually employ some kind of non-linear optimization to fit the model of the hand to image evidence, the real-time performance requirement is in general not satisfiable. In our setting, however, this in itself would not be a limiting factor as the dimensionality of our problem is much smaller than the full 27D space. Unfortunately, another problem would surface: as they are dependent on temporal coherence to avoid lost tracking situations, abrupt hand-state changes (like posture-change or rapid hand movement) are beyond the reach of such methods. According to [Stu92] this problem can be alleviated by using imaging frame rates of at least 100Hz (in contrast to the usual 25-30Hz), which results in new problems: a) expensive special hardware, b) the time available for processing one frame reduces drastically, c) if the full 27DOF are not explicitly modeled, the system cannot interpret transient states between postures. The major drawback of these methods from our point of view is that they require special initialization procedures (the state of the hand in the first frame should be available) and they cannot handle automatic lost detection. Another question regarding these methods is how much they depend on the similarity between the hand model and the user's hand.

5.2.2 Single Frame Pose Estimation

First, we want to mention that in principal every model-based approach can also be used for single frame pose estimation by testing all possible hand configurations. However, such an approach would in general be too slow for interaction purposes. Therefore, model based approaches typically aim at reducing the search space or massive parallel computing. For instance [GAW+06] use a particle based model fitting approach, but needed a PC cluster of 9 cluster nodes to achieve 10 frames per second. Moreover, fast hand movements needed up to 9 frames in order to achieve convergence of the model to the correct pose/posture. In [CUK+08] the coarse 3D shape is constructed from 4 camera images by using the shape-from-silhouette technique (see Sec. 2.1). Then, based on an initial estimate a hand model is fitted to the data. In [BEM07], using the range data obtained by a Time-of-Flight camera a first estimate of the coarse hand position and orientation based on PCA is computed. Then, a model based approach is used to estimate the full hand configuration. However, both approaches does not have real-time capabilities and suffer from self-occlusions of the hand.

Therefore, object detection approaches processing independent frames commonly adopt appearance models [VJ04], [KT04b]. A learned appearance model can efficiently be detected every new frame, so these methods do not suffer from

the initialization problem. Unfortunately, because of the usually employed appearance models, they have the disadvantage that out-of-plane rotations cannot be handled. As a consequence, their typical application is to implement posture recognition systems; the user's palm should in general be parallel to the image plane. To overcome this restriction, multi-view systems with best view selection algorithms have been proposed [UO99]. Nevertheless, these systems suffer from angle restrictions.

[MZ09] presented a novel method for real-time detection of articulated objects in images, which can be applied for hand pose estimation. Therefore, a set of template images is constructed in a preprocessing step and the detection is solved as a database query. The key ingredients of this approach are a novel edge gradient operator to generate edge images and the formulation as a convolution, which can be computed very efficiently in Fourier space. For a restricted template database this approach showed promising results.

5.3 Partial Pose Estimation

A common approach for solving the problems of full DOF tracking such as initialization or low performance is restricting the tracking to a subset of hand DOFs. Commonly only the global position and orientation of the hand is recognized and several postures are distinguished. In the particular field of posture recognition many articles exist which involve machine learning techniques and global feature extraction methods. As posture classification is not the focus of our work we refer the interested reader to e.g. [vHB01, ASO00, OSK02, MA07]. The existing approaches for partial pose estimation can be classified according to whether they extract 2D features or 3D features of the hand.

5.3.1 2D Features

For estimating the 2D position of the hand from an image one choice is computing the center of gravity of the hand segmentation mask (e.g. [SSK01, JKS98]). A solution considered to be more stable is computing the point having the maximum distance to the boundary of the hand silhouette [UO99, KSK01, MLN05, ASO00]. [ASO00] also determines the 3D hand position and recognizes 15 distinct handpostures by using a two-camera system. One of the cameras has a side-view, the other observes the user's hand from above. The hand of the user has to be parallel to the image plane of the top camera.

[UO99] (based on [UMKN96]) presented a 5-camera system with automatic view selection for pose and posture recognition. The system was able to track both hands of the user in seven postures. The 3D hand position is determined

by combining the 2D centers of gravity and the 3D hand orientation by using 2D edge information from the images. However, the system suffered from restricted angles of rotation.

Approaches that track more than the hand position commonly aim at identifying the fingertips. Using the fingertip locations the hand position and orientation as well as the fingertip orientations can be computed more easily. Different solutions for fingertip detection exist. One approach is using correlation techniques in a circular mask [KSK01, OSK02, LB04]. All three systems used one or two cameras from above to track the 2D positions and orientations of the fingertips and the hand. Therefore, out of plane rotations can not be handled and the 3D hand orientation is not recovered. The systems are already capable of simultaneously tracking both human hands.

In contrast, [OZ97, CBC95] solved the task of fingertip detection by using fingertip templates extracted from real images. Both systems use one camera from above, thus, they are only able to recover 2D positions and orientations and can not handle out of plane rotations.

Several approaches [SK98, ML04, OZR02] extract curvature maxima at the boundary of the segmentation masks to identify fingertips. In [SK98] four hand-postures are recognized and 5DOF of the hand are tracked (position as well as elevation and roll of the hand) with a stereo vision system. The tracked elevation and roll angle-ranges are limited based on the camera placement to $\pm 40^\circ$ and $\pm 30^\circ$, respectively. In [SK99] a similar system has been developed with monocular vision and almost the same capabilities. [ML04] introduced the Visual Touchpad. Two cameras from above track the 3D hand position and 2D orientation as well as the fingertips of both hands simultaneously. [OZR02] introduced a stereo tracking system for 6DOF pose tracking in 5 hand postures. The effective ranges of the rotation angles were limited and also dependent on the tracked posture.

Last but not least, fingertips can also be extracted by computing contour points having a maximal distance to the hand position [MLN05, JKS98, SSK01]. In [MLN05] one camera is used for tracking the 2D positions of one hand and its fingertips. Instead, in [JKS98] two cameras are used to track the 3D positions of both hands and their fingertips. However, the method still suffers from angle limitations. In [SSK01] the center of gravity is used as the hand position and a 3-layer neural network is used to distinguish between 6 possible postures.

Having computed the positions of the fingertips, the fingertip orientations can be approximated by the directions of the principal axes of a window around the fingertip position (e.g. [UO99, SK98]).

While most of these methods are able to process independent frames (i.e. they do not need an initialization procedure), the coherence can additionally be exploited to gain computational speed and robustness by using Kalman filters (e.g. [MLN05]) or heuristics for defining search windows in the images (e.g.

[OZR02]).

5.3.2 3D Features

Other approaches track features directly in 3D. In [Jen99] different features like edges and convexities are extracted both from stereo range images and color images. These features are then combined to obtain the 3D position and orientation of one finger.

In [DS99], 3D cylindrical fingertip models are fitted to 2D image data in order to obtain finger positions and directions. However, out of plane rotations or cluttered backgrounds can not be handled.

In [GBCB00] a stereo camera pair is employed to obtain range data as well as color data. The 3D hand position is obtained by computing the centroid of the range data belonging to the hand and arm. To compute the hand orientation, a 3D line and a 3D plane are fitted to the range data to obtain the arm direction and the hand palm normal. Subsequently, the color image of the hand is used for gesture classification. However, as the orientation is determined by the arm direction, bending the wrist joint can lead to involuntary pose changes. Moreover, fitting a plane to extract the hand palm is problematic when the hand does not form a flat posture. In [MS08] a similar line and plane fitting algorithm is used, but instead of a stereo camera system a Time-of-Flight (ToF) camera is used. Furthermore, a hand model is matched with the data to obtain the full hand configuration. Further problems of both approaches are self- occlusions because the hand is observed only from one direction.

[GLA+08] and [SPHK08] use a ToF camera in order to estimate the center of mass as the 3D hand position as well as to recognize several postures exploiting an appearance based approach. [GLA+08] used a low resolution ToF camera (64×48 pixels) in combination with a VGA gray scale camera. [SPHK08] employed a higher resolution ToF camera (176×144 pixels) and therefore do not need an additional camera.

[BPS+09] use a ToF camera, which additionally captures RGB information to first extract finger features of the hand. Then, a 3D hand model is fitted to the data to obtain the joint angles of the fingers. However, the full hand orientation could not be recognized and only limited finger movements could be tracked correctly.

In the field of ToF camera-based hand-tracking also other approaches exist that are directly developed for commercial use and therefore not published (e.g. Mgestyk, http://www.mgestyk.com). Therefore, the exact mode of operation and capabilities are not known. However, as all these existing approaches use only one depth camera, the estimation of the full global hand orientation without angle limitations is probably not achieved.

In contrast to the full DOF methods outlined in Sec. 5.2 the partial pose estimation approaches are all capable of real-time performance and do not need any special initialization procedures. Although some of them support more postures than our system, none of them is capable to track all the 3 rotational DOFs without limitations. This is due to the fact that either they rely on feature extraction in 2D and combine the extracted features to obtain the 3D pose or they capture the hand only from one direction, whether 2D, depth or stereo camera. Combining all the silhouette information from different views into the visual hull and conducting the feature extraction in 3D allows our method to overcome the limitations present in other systems.

Part II

Interaction Techniques

CHAPTER 6

BACKGROUND

6.1 Exploiting Hand Movements for Interaction

6.1.1 The Pose

The pose of one hand is typically described by a 3D position vector and a 3D orientation quaternion both updated in each frame when the hand-tracking procedure succeeded. Having tracked both hands two hand poses are obtained, which can be exploited for various interaction purposes. Up to 12 continuous degrees of freedom can simultaneously be controlled via both hands. For example, the pose of each hand can directly be exploited for interaction by mapping it to the pose of a virtual entity (e.g. a rigid 3D object) in a virtual environment. Or a combination of the hand poses can be used for symmetric object manipulations (a virtual object is moved as if gripped between both hands). Furthermore, only subsets of the hand poses' DOFs can be exploited for e.g. simulating a 2D mouse or joystick. As long as directional compliance of real and virtual movements (i.e. movement direction is corresponding) is maintained, these kinds of interaction can be very intuitive.

6.1.2 Postures

The hand-tracking method we developed is capable of recognizing four different stiff postures (see Table 3.1) per hand. An interaction interface typically exploits different postures for simulating different (button) states. For example using one posture for a button up state and another for a button down state to simulate clicking events or using different postures for changing the current mode of action (e.g. one posture for manipulation and another for selection).

Postures offer a simple way of compensating the absence of physical buttons. However, two major drawbacks lead to a limited utilizability. First, posture changes mostly induce involuntary changes to the hand pose, because it is nearly

impossible to move one or more fingers without inducing additional tremor to the whole hand. Moreover, the pose is unintentionally influenced during a posture change due to the dependency of the tracked hand pose on the positions of the fingertips. Therefore, the exploitation of postures for e.g. releasing a virtual object at a precise pose (3D manipulation) or clicking on a precise position (virtual pointer selection) is not reasonable.

Second, the use of postures can be very demanding for the user's fine motor skills due to the complexity of finger movements. The needed strain and concentration is considerably higher if posture changes have to be performed. Using several different postures in one application can additionally be very confusing. We argue that exploiting postures can be helpful in some cases, however, simply using postures as a button adequate is not advisable.

6.1.3 Gestures

In this thesis a gesture is defined as a specified translational and/or rotational movement of the hand(s). Thereby, such a movement can be specified in various ways as for example "one hand moves faster than a certain velocity" or "the right hand performs a circular movement in the xy-plane". The recognition of gestures is not part of the tracking algorithm and should instead be integrated into the interaction interface. Typically a set of predefined gestures is available in an interface and each gesture can be assigned to a specific command in the current application.

One drawback of gesture exploitation is that the hand movements corresponding to the gesture are not longer available for tasks other than the assigned command. Additionally, the user has to ensure by herself/himself that she/he only performs the gesture if her/his intention is issuing the assigned command. On the contrary, gesture exploitation does not suffer from the problems that posture exploitation does. In particular, performing gestures can be simpler and more comfortable than switching between different postures. Furthermore, as gestures are not defined and restricted by the hand-tracking procedure they can be purpose-designed for certain applications or tasks. We argue that gestures must be integrated very carefully, but then they can serve as an extremely powerful method for designing efficient and easy-to-use interaction.

6.2 Basic Interaction Techniques

In this section we outline some existing interaction techniques serving as a basis for the interaction techniques that will be introduced in the following chapters.

Moreover, we shortly discuss the specific problems and limitations in their use for bare-handed interaction.

6.2.1 Virtual Hand

One category of fundamental 3D manipulation techniques are virtual hand techniques, where the user's hand(s) can directly be used to select and manipulate a virtual object. Therefore, the user's hand motion is directly mapped to the motion of a virtual 3D model (typically a model of the human hand(s)) acting as a cursor. Then, virtual objects can be selected by intersecting the cursor with the target of selection and performing a predefined selection command (e.g. voice command or specified hand posture). Subsequently, the movements of the selected object are coupled to the movements of the virtual hand. This way, selection and manipulation of virtual 3D objects is intuitive as it simulates human interaction in everyday life.

However, such techniques suffer from the problem of a limited work space. Only objects within the user's reach can be selected and a selected object can only be manipulated inside this region. To enable selection and manipulation outside the user's reach her/his virtual reference frame (defined by the virtual camera pose) has to be changed. In this context, several approaches were introduced as for instance the World-in-Miniature technique [SCP95], HOMER (hand-centered object manipulation extending ray-casting) [BH97], Scaled-World Grab [MFPBS97] or Voodoo Dolls [PSP99]. However, these techniques always include the need of switching between the selection/manipulation mode and another mode to travel in or scale the virtual environment. This turned out to be inconvenient and difficult to handle in many settings.

Other techniques simply aim at enlarging the user's reach. For example the Go-Go technique [PBWI96] simulates an interactively non-linear growing of the user's arm(s). When a user's hand is close to the user, the mapping from the real hand pose to the virtual object pose is one to one. As she/he extends her/his hand and arm beyond a certain range, the mapping becomes nonlinear and the virtual arm "grows". Thus, she/he is able to reach objects further away. However, this technique decreases the precision for selection and manipulation of farther objects. Furthermore, the problem is only shifted as the working space is simply enlarged.

Another approach is PRISM [FK05], which helps to increase the accuracy of object movements. This interaction technique acts on the user's behavior in the environment to determine whether they have precise or imprecise goals in mind. When precision is desired, PRISM dynamically adjusts the control-to-display-ratio which determines the relationship between physical hand movements and the motion of the controlled virtual object. A similar approach to automatically

adjust the speed of the current action is described in [Osa06]. These techniques can enlarge the user's reach without loosing precision. However, they only shift the problem to a larger scale and do not completely solve it.

6.2.2 Virtual Pointer

Another class of fundamental 3D selection and manipulation techniques are pointing techniques. Selection is performed by intersecting the ray of pointing with a virtual object or item (see Fig. 6.1) and issuing a selection command. This sup-

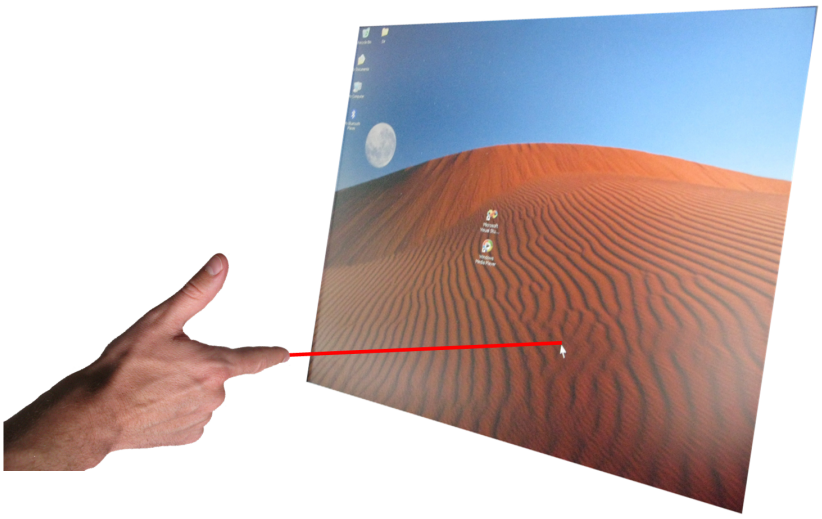


Figure 6.1: The virtual pointer. Cursor position is determined by the pointing direction of the hand.

ports easy selection of virtual objects located beyond the user's reach. For manipulation purposes the object can be attached to the end of the pointing vector after such a selection operation.

The virtual pointer technique is the most commonly used method for selection. Various experimental evaluations demonstrated a superior selection performance than virtual hand techniques due the need of less hand movement (according to [PIWB98]). However, pointing is badly suited for manipulation; positioning is only possible radially around the user and rotating only around the pointing direction.

Another problem of virtual pointer techniques is the accuracy of cursor positioning in selection tasks. As the cursor position is determined by the pointing direction of the hand the selection of small objects or items can be cumbersome due to natural hand tremor. The problem even intensifies if a high resolution display is employed or the hand-tracking accuracy is low. To this end, techniques

for increasing the cursor precision are needed, which do not or only minimally interfere with the intuitiveness and ease of use of the positioning task.

6.2.3 Bi-manual Symmetric Manipulation

Two-handed 3D interaction techniques are commonly distinguished (according to [Gui87]) into bi-manual asymmetric, where the hands perform different actions (e.g. the non-dominant hand holds something while the other manipulates it), and bi-manual symmetric, where the hands perform identical actions (e.g. pulling a rope, typing on the keyboard). In our research we focused on bi-manual symmetric techniques.

In the context of symmetric manipulation Cutler et al. [CFH97] introduced several techniques: the grab-and-twirl technique enables the user to pick up two sides of an object and then to carry and turn it around with both hands. By fixing one or more DOFs of the applied transformation several other techniques were derived such as the grab-and-carry technique (no roll around the line connecting the two hands is allowed) or the turntable technique (only turning around a fixed axis of rotation is allowed). These techniques turned out to be both intuitive and efficient. However, they implicitly assume the employment of the virtual hand metaphor for determining the two pivot points (i.e. the two points where the object is grabbed). Overcoming this restriction was part of our research.

CHAPTER 7

SINGLE-HANDED TECHNIQUES

The system described in Sec. 3 enables real-time tracking the position and orientation of the user's hand in different postures. This allows for using the human hand as a natural input device. However, the absence of physical buttons for performing click actions and state changes poses severe challenges in designing an efficient and easy to use 3D interface on top of such a device. In particular, solutions have to be found for clicking menu items, selecting objects and coupling and decoupling the object's movements to the user's hand (i.e. grabbing and releasing). To this end, we introduce a novel visual feedback in order to support the ease of using this device, a novel superior technique for grabbing and releasing objects together with a user study, an efficient clicking operation for selection purposes and a method for improving cursor accuracy. We further describe how virtual camera steering can be improved and how switching between different modes of interaction can suitably be performed in a 3D interface.

7.1 Visual Feedback Box

Visual feedback comprises every visual response in a GUI to an action performed by the user with an input device. In order to support the user's handling of an interface, we integrate specific visual feedback into our GUI, which is shown in a small window that can be positioned freely (typically in the upper left or right corner). The visual feedback provides three basic visual cues (see Fig. 7.1):

1. A hand model, which is moved according to the user's hand in order to indicate the recognized pose and posture.

2. A cuboid, in order to show the working volume of the hand-tracking system. If the user moves her/his hand inside the working volume, the hand model (see Cue 1) is shown inside this box. This way the user gets a visual hint, if her/his hand can be seen by enough cameras and if the tracking is working correctly.

3. The shadow of the hand model on the floor of the cuboid. This helps the

user to estimate the position of the hand along the z-axis more accurately, especially if currently no 3D imaging is adopted.

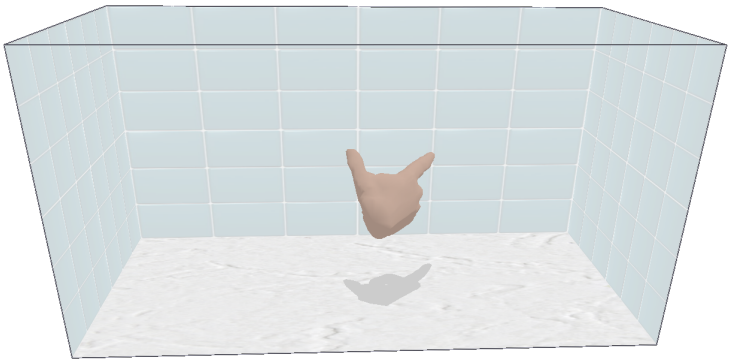


Figure 7.1: An example of the basic visual feedback.

Furthermore, the visual feedback can be extended with other visual cues in order to help users to learn how to interact in the current mode of action (e.g. selection or manipulation). Currently, there are three major extensions for the respective modes of interaction, which will be explained in the following respective sections. Here, we just want to mention that in an informal user study several subjects could practice with a 3D interface comprising these mode of interaction. After that, we questioned them for their subjective impression concerning the visual feedback with its respective modifications. Most subjects told us that the proposed visual feedback for different interaction modes helped them notably for familiarizing with the respective interaction mode and for ensuring that their handling of the hand-tracking device is correct (e.g. whether the hand is still in the working volume and forms the correct posture).

Note that more objective testing of the visual feedback goes beyond the scope of our work, because as it supports the familiarization with an interface, we can not compare the performance with and without it for one and the same subject. However, the timings and precision strongly depends on the individual, so an objective study would need a great many of subjects.

7.2 Jerky Release

Once an object is selected and the full amount of the 6 continuous DOFs of the global hand pose is employed for moving the object, an additional suitable mechanism is needed for determining when the object shall be attached to the hand or not. This mechanism should enable precise releasing of objects and should be manageable fast and efficiently. To this end, we used an approach based on the

velocity and acceleration of the hand translation and rotation to trigger grabbing (attach action) and releasing (detach action). The idea is to move a virtual object only as long as the user moves her/his hand smoothly and performs no abrupt pose changes. If she/he instead performs a fast and jerky movement the object is released. This provides an intuitive interaction metaphor as it corresponds to real life experience (e.g. if a screw is turned downward, people typically do a relatively slow clockwise rotation while turning the screw and a relatively fast counterclockwise rotation back without turning the screw).

This behavior is implemented in the state machine depicted in Fig. 7.2. C_a and C_v are conditions based on the translational and rotational velocities and accelerations v_t, v_r, a_t and a_r and are defined as

$$\begin{aligned} C_a &= a_t > A_t \vee a_r > A_r, \\ C_v &= v_t < V_t \wedge v_r < V_r, \end{aligned}$$

where V_t, V_r, A_t and A_r denote the respective thresholds. Note that by condition-

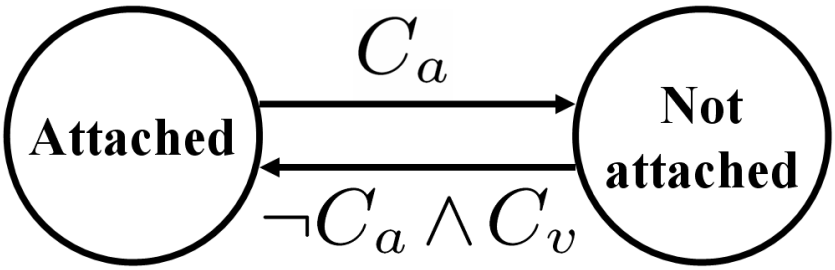


Figure 7.2: State machine illustrating how an object can be grabbed and released.

ing the signed accelerations no releasing is performed, when the user jerks to a halt (leads to high negative accelerations). In our current setting the thresholds were set to be $A_t = 50\frac{cm}{s^2}$, $A_r = \frac{5}{3}\pi\frac{rad}{s^2}$, $V_t = 40\frac{cm}{s}$ and $V_r = \pi\frac{rad}{s}$. These threshold values were determined in a pilot experiment, where first an object should be moved in only one direction (we used the positive x-direction for translation and a clockwise rotation around the roll-axis for rotation) with different employed threshold values while the percentage of involuntary movements was measured. A performed hand movement is classified to be *involuntary* either if the movement is toward the desired direction but the object is not moved (the acceleration threshold was exceeded without intension) or if the movement is toward the opposite of the desired direction but the object is moved (the acceleration threshold was not exceeded). Note that using lower threshold values leads to more occurrences of the first kind of involuntary movements while higher threshold values abets the second kind. Therefore, we rate the chosen thresholds by the sum of the squared respective errors (the amount of involuntary movements). This way, both kinds of

involuntary movements are minimized. Additionally, we let the subjects perform some simple manipulation tasks while the completion times and precision were measured (similar to user experiments which will be described in Sec. 7.2.1). We noticed a strong coherence between good performance (completion times and precision) and good threshold rating. The above stated thresholds result from several of these tests and had the best overall performance. In our interface, the user can adjust these values by choosing a factor from the interval $[0.5; 2]$ with which all threshold values are scaled in order to account for the different preferences of each individual. However, for the user experiments in Sec. 7.2.1 this factor was locked to 1.

When a fast and jerky movement is performed, typically the acceleration curve has a positive peak at the beginning, then decreases to approximately zero and has a negative peak in the end (see Fig. 7.3(Right)). Therefore, the condition C_v is essential for conditioning a transition back to state 'Attached'; otherwise the 'Not attached' state would be left directly after the positive peak at the beginning.

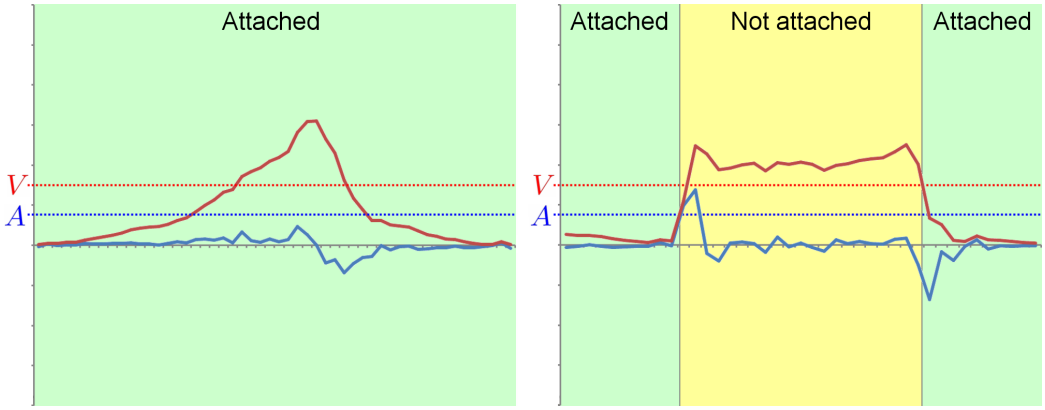


Figure 7.3: Diagrams of velocity (red) and acceleration (blue) across several frames (x-axis) of two different translational movements. The two horizontal dashed lines indicate our chosen thresholds A_t and V_t and the green and yellow regions the periods of having attached or released the object. Left: A movement is performed by smoothly increasing the speed. Right: A fast and jerky movement is performed.

With this implementation a jerky translational or rotational hand movement induces a transition from state 'Attached' to state 'Not attached' and if the translational and rotational speed as well as the acceleration of the hand movement falls below the according given thresholds a transition back to state 'Attached' is performed. As long as the speed is not abruptly increased the object will move according to the users hand.

Note that if a simple thresholding on the velocity would be used, precise releasing of the object would hardly be possible, because dependent on the acceleration it could take several frames until the velocity threshold is reached. But in these frames the object would still be moved. Using a very low velocity threshold to diminish this problem would prohibit the user from performing any fast operation. This can be seen in Fig. 7.3, where we depicted graphs of velocity and acceleration for two translational motion sequences: one, where the user performed a typical smooth manipulation (Left), and another, where she/he performed a fast and jerky movement (Right). Analogously, we recorded two rotational motion sequences, which showed the same characteristics.

In addition, using the acceleration as a single criterion for transiting to state 'Not attached' has the advantage, that this way, the velocity with which an object can be moved is not constrained. For example in the motion sequence of Fig. 7.3(Left) the object was attached all along although the velocity exceeded the velocity threshold.

Using the acceleration criterion for releasing operations enables fairly precise positioning of objects. However, sometimes slight unintentional movements occur in the direction the fast and jerky movement is performed, because dependent on the jerkiness of the performed movement it can take a short while until the acceleration threshold is exceeded. To overcome this problem, we introduce a simple post-correction step, when the 'Not attached' state is reached in frame i. We undo the last k manipulation steps (both translation and rotation), whereby k denotes the greatest number of steps, that fulfill the following conditions for all j with $i - k \leq j < i$:

$$\left(\frac{\|p'^j\|}{t^j - t^{j-1}} < \frac{\|p'^{j+1}\|}{t^{j+1} - t^j} \right) \wedge (t^i - t^j) < t^0. \tag{7.1}$$

t^j denotes the time and p'^j is defined depending on whether the 'Not attached' state is reached due to a high translational or high rotational acceleration. In the case the translational acceleration threshold is exceeded, p'^j is defined to be the hand position increment $p^j - p^{j-1}$ (p^j denotes the hand position in frame j). In the other case, we instead define p'^j to be the normalized rotation axis of quaternion q'^j multiplied by its angle. q'^j is defined as the rotation of the hand from frame $j-1$ to frame j. The first condition in Eq. (7.1) ensures the absolute acceleration value to be strictly increasing for the k steps. This avoids unintended undoing of steps, if for example the user moves an object, stops shortly and then wants to release it by performing a fast and jerky movement. The second condition prohibits undoing steps that are longer ago than t^0. This threshold describes the maximal available time the user has to exceed the acceleration threshold. In our current setting t^0 is chosen to be 100 milliseconds as several experiments showed this to be suitable. This post-correction enables very precise release operations in

all manipulation tasks. Moreover, because only very slight post-corrections are needed, the distraction induced by automatic undoing is only marginal.

The visual feedback for this technique shows how an object is moved according to the position and orientation of the user's hand. A small solid cube is elastically attached to the hand model. The elastic relationship between the hand model and the object inherently indicates that the object will be released if the hand moves too fast. If the object is released in the current task, the cube is released in its current pose (see Fig. 7.4(Middle)). If the object is grabbed in the current task, the cube jumps back to the hand model and is reattached (see Fig. 7.4(Right)). This indicates that an object will be released if the hand moves rapidly.

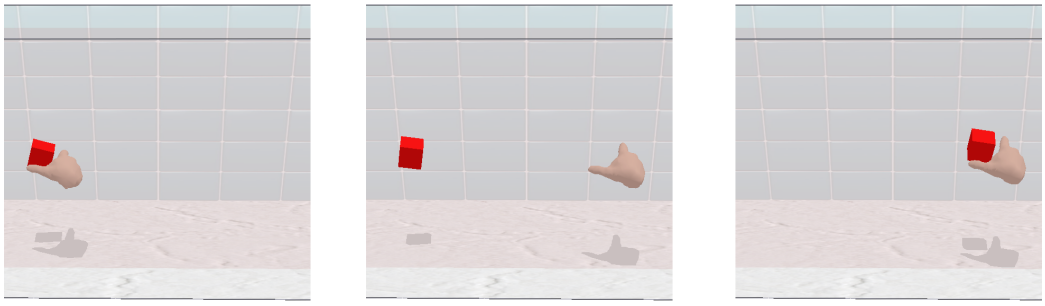


Figure 7.4: Illustration of a grab and release cycle. The hand model is rendered together with the red cube. Left: The feedback when the object is attached to the hand. Middle: An abrupt hand movement toward the right is performed. Therefore the cube stays in the pose where the abrupt movement started. Right: The hand grabs the object again when it moves slowly. State 'Attached' is again reached. The cube jumps back to the hand model.

7.2.1 Experiments

We conducted an experiment to evaluate the performance of the proposed technique for grabbing and releasing virtual objects. Our hypothesis was that our method would be superior to other techniques that are commonly applied for hand-tracking devices. Moreover, we expected that a hand-tracking device combined with our technique had superior or at least similar performance compared to a standard 6 DOF controller.

To this end, we compared our technique to both the use of a grabbing posture (i.e. only if the subjects form the grabbing posture the virtual object moves according to her/his hand) and the use of a standard 3D mouse. Several simple manipulation tasks had to be solved by using each of the controller types, while the completion times and precisions were measured.

For the posture based grabbing and releasing the adjustments described in [Osa06] for precise releasing were implemented, where the release pose of a virtual object is automatically adjusted based on the relative speed of the two grabbing fingers (the thumb and index finger). However, we could not use the same velocity threshold of $1\frac{cm}{s}$ as proposed in [Osa06], because depending on the velocity of the hand pose the accuracy of the determined thumb and front-fingertip positions were not sufficient (Osawa used data gloves, which have a high precision and whose global pose does not influence other hand parameters). Therefore, we had to use a higher threshold ($10\frac{cm}{s}$ in the experiments), which led to inferior releasing precisions.

Experimental Setting

For running the virtual environment application a second computer (Intel E6600, Geforce 8800 GTX) was connected to the hand-tracking computer (see Sec. 2.5). The application was visualized on a standard 19" TFT-Display. Additionally, a 3D connexion SpaceNavigator was connected to this PC.

Experimental Tasks: In the experimental tasks a virtual object had to be approximately moved to a specified position (less than 2 units translational error) and/or orientation (less than 4 degrees rotational error) by using the different techniques/controllers. In the first task only translation had to and could be modified until the desired position was approximately reached. In the second task, the object's position was fixed and only orientation had to be modified. These two tasks were established in order to check if one of the techniques has specific advantages in either the rotational or translational DOFs. To check the performance for more complex tasks the orientation and position had to be manipulated simultaneously in the third task. These three tasks were used to measure the completion times.

In the fourth and last task again orientation and position had to be modified, but the subjects could decide by themselves when the final pose was reached and then had to release the object by either pulling the hand out of the working volume (if the hand-tracking device was used) or pressing the left 3D mouse button (if the 3D mouse was used). No snapping algorithm (the object snaps to the desired pose, when it is near by) was applied. This task was used to determine the positioning errors.

Participants: Eight participants (one female, seven males, all university students) took part in the experiment. They had little or no virtual reality experience.

Procedure: Each participant had to solve all four tasks four times by employing each of the three controllers (3D mouse, hand-tracking with grabbing posture and hand-tracking with our technique). Thereby, the sequence of the employed controllers was permuted evenly and all tasks had to be finished until the next controller was adopted. Before starting the test for each controller, its mode of action was explained and the subjects could familiarize with it in a short preparation time (two minutes).

Results

The average task completion times for all individual subject are depicted in Table 7.1 and the total average task completion times including standard deviations are illustrated in Fig. 7.5.

Subject	**Grabbing Posture**			**Our Technique**			**3D Mouse**		
	T	R	TR	T	R	TR	T	R	TR
1	21.3	14.7	19.4	12.3	11.6	17.2	10.4	6.7	16.2
2	23.6	18.3	39.7	9.5	8.7	13.7	8.9	4.5	22.4
3	25.9	35.0	75.6	15.6	22.8	28.9	20.0	20.1	64.0
4	15.8	10.9	47.1	9.2	7.3	24.6	4.7	5.1	15.5
5	23.0	79.3	84.1	14.5	21.1	19.0	17.1	10.0	25.7
6	11.8	31.3	46.7	10.0	5.1	19.4	6.9	10.5	12.9
7	23.2	35.3	57.3	13.4	15.1	37.2	16.5	23.8	42.8
8	17.2	27.9	46.9	10.5	6.1	14.9	3.9	2.4	18.8
Average	20.2	31.6	52.1	11.9	12.2	21.9	11.0	10.4	27.3

Table 7.1: Mean task completion times (in seconds) for all individual subjects in the following sub tasks: translation (T), rotation (R), translation and rotation (TR).

Employing the grabbing posture was clearly inferior to our technique or the 3D mouse. This is mainly because some time is needed for switching the postures. If the object's orientation is manipulated, this becomes even more relevant, because more grab and release cycles are needed due to the little space of anatomical rotational freedom.

Considering both the times of our technique and the 3D mouse it can be seen that the 3D mouse performs slightly better if the amount of degrees of freedoms is restricted, but inferior if all 6 DOFs are available. This was confirmed in our observations during the experiments. The simultaneous control of several DOFs was significantly more difficult with the 3D mouse.

In Table 7.2 the individual mean errors and in Fig. 7.6 the total mean errors and their standard deviations for translational and rotational positioning of the virtual object are depicted.

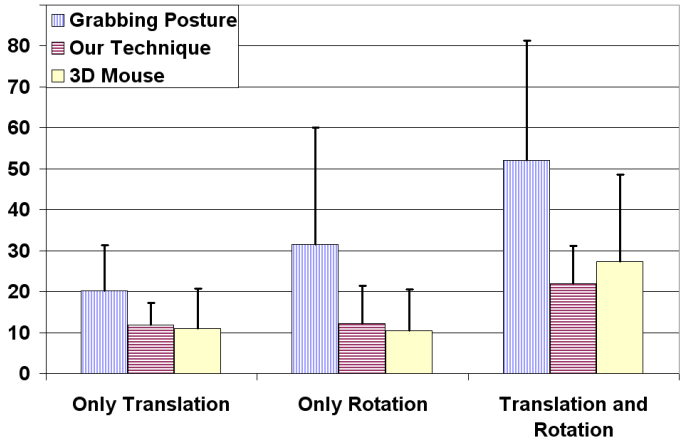


Figure 7.5: Task completion times (in seconds) with standard deviations (error bars).

The diagram illustrates, that our technique is suitable for precisely releasing virtual objects. The bad results for using the grabbing posture are caused by overhasty releasing while the hand was still attached to the virtual object.

In general, for handling object movement by using the Jerky Release technique, most users needed a short adaptation phase until they developed a sense for the different kinds of motion (smooth movements for moving the object and fast/jerky for releasing it). But subsequently, they could easily perform different complex tasks.

Obviously, a limitation of the grabbing and releasing technique is the fact that a virtual object can not be moved fast and jerky any more. However, in practice such movements are utilized rarely for manipulation tasks. To quantify this problem, we analyzed the movements of both hand and virtual object in our user experiments for the cases that the grabbing posture was employed for grabbing and releasing instead of our technique. We computed the percentage of virtual object movement that occurred while the 'Not attached' state would have been occupied, if our technique would have been used. On average, less than 5% of the virtual object movements would have been filtered out by our technique. Moreover, we observed such movements often to be unintended by the subjects, for example if the virtual object should be released by switching from the grabbing to the standard posture but moving the hand overhasty while the object is still attached to the hand.

Subject	Grabbing Posture		Our Technique		3D Mouse	
	T	R	T	R	T	R
1	1.1	2.5	0.8	2.8	1.1	2.5
2	0.9	2.2	1.0	2.8	1.0	3.0
3	1.7	4.4	1.2	1.9	1.0	2.5
4	1.0	2.6	1.2	2.2	1.1	2.6
5	2.6	3.4	0.9	2.1	1.2	1.8
6	1.2	2.3	0.9	2.1	0.9	2.5
7	1.5	3.2	1.2	2.6	1.0	2.4
8	1.4	2.7	1.0	2.8	1.0	2.6
Average	1.4	2.9	1.0	2.4	1.0	2.5

Table 7.2: The mean errors for all individual subjects separated into translational (T) and rotational (R) error. In degrees for the rotational error. The translational error can only serve as a relative measure as it depends on the adopted mapping from real to virtual space.

7.3 Roll Click

For the selection of objects and menu items a clicking mechanism is needed to trigger button events. Thereby, the clicking mechanism should be easy to learn as well as easy to perform. To this end, we decided to exploit one DOF for triggering button events. We found exploiting a specified rotation around the roll-axis (i.e. around the axis described by the forearm) serves best. This has two major reasons. First, exploiting a rotational DOF for clicking is superior to using a translational DOF due to comfort issues. Using a translational DOF would force the user to move the whole fore arm in order to perform a click.

Second, a rotation around the roll-axis performs better than around the yaw or pitch-axis, because the range of rotation the user can utilize for this rotation is significantly larger. Moreover the roll-angle's value is only marginally affected by changing the hand's position or pointing direction. The yaw and pitch angles depend loosely on the position of the hand (e.g. translating the hand toward the left induces a rotation toward the left except the wrist is bended for compensation), which could lead to unmeant clicking operations.

Our approach is particularly advantageous compared to exploiting a posture or the second hand for clicking because it is significantly easier and less exhausting to perform. We observed some users to be nearly incapable to switch between specified postures while further concentrating on the current task.

For deciding if a virtual button event is triggered in frame i (the i-th time the hand pose/posture was determined), we use two sufficient conditions based on the value of the user's hand's roll-axis angle α_r^i. The first condition enables very slow clicking with a more spacious movement while the second condition

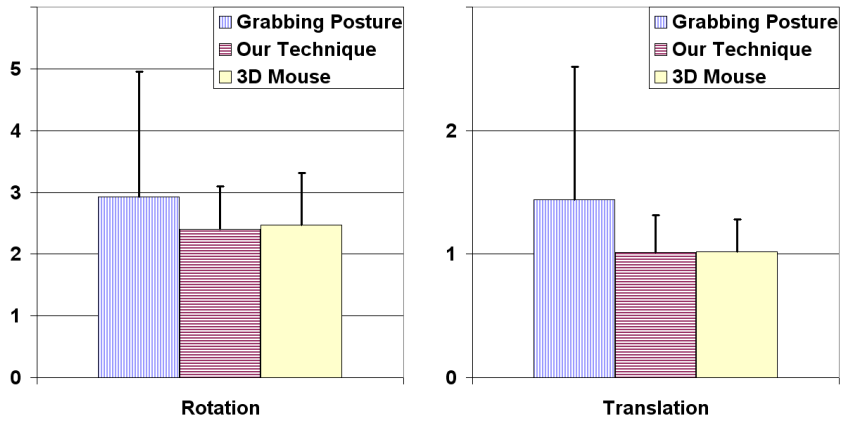


Figure 7.6: Positioning errors with standard deviations (error bars). In degrees for the rotational error. The translational error can only serve as a relative measure as it depends on the adopted mapping from real to virtual space.

also enables clicking by smaller but faster movements. Note that one condition would be sufficient, but using both conditions better accounts for the individual user preferences. The first condition employs a hysteresis thresholding (i.e. a thresholding, that employs different threshold values depending on the state that is occupied) based on α_r^i for triggering a button event. This is expressed in the first terms in Fig. 7.7, respectively. T_1 and T_2 denote the hysteresis thresholds. Currently we use $T_1 = \frac{\pi}{4}$, $T_2 = \frac{\pi}{16}$. In our current setting the roll-axis angle is defined to be zero, if the index finger is pointing toward the front and the thumb is pointing up. A counterclockwise rotation of the user's hand around its roll-axis increases this angle while a clockwise rotation leads to a decrease.

The second sufficient condition is expressed in the second terms in Fig. 7.7, respectively. These terms comprise one of the conditions C_L or C_R and an additional constraint based on the hysteresis thresholds. The additional constraints are needed to disambiguate between left and right button events; otherwise a right button down event could not be distinguished from a left button up event. The conditions C_L and C_R are based on α_r^i as well as on the signed angular velocity v_r^i of the roll-axis angle, which is defined as

$$v_r^i = \frac{\alpha_r^i - \alpha_r^{i-1}}{t^i - t^{i-1}}, \tag{7.2}$$

where t^i is the time of frame i. Now C_L can be defined as follows. If k is the greatest positive number with $v_r^{i-j} \geq \epsilon$ for all $j = 1, ..., k$, then the condition C_L is defined as

$$C_L = v_r^i < \epsilon \ \wedge \ (\alpha_r^{i-1} - \alpha_r^{i-k} > \frac{\pi}{16}). \tag{7.3}$$

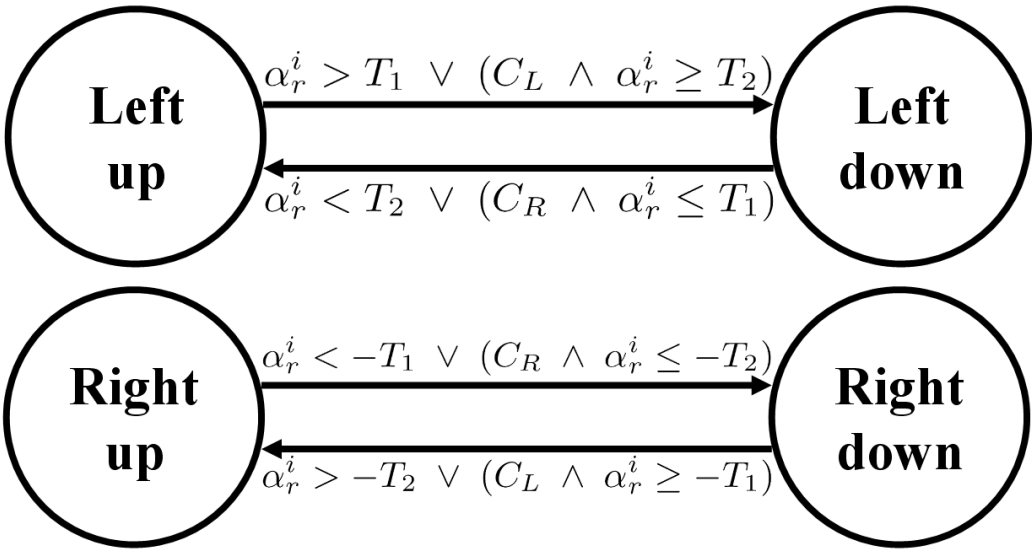


Figure 7.7: State machines illustrating when the virtual left and right button are pressed or released, respectively.

This way, already a small counterclockwise rotation can be employed to simulate a left button down or right button up event. The threshold ϵ ensures the rotation to have a minimal velocity (we used $\epsilon = \frac{1}{2}\pi\frac{rad}{s^2}$). The second term of Eq. (7.3) is needed to avoid unmeant button events by requesting the angular movement to exceed a minimal value (otherwise an infinitesimal movement could lead to a button event). C_R is defined analogously by substituting $-\epsilon$ for ϵ and $-\frac{\pi}{16}$ for $\frac{\pi}{16}$ and inverting the relational operators.

If a clicking operation is performed, we observed the pointing direction to lack accuracy due to unmeant angular movements around the pitch or yaw-axis. Therefore, selecting a small object by employing the virtual pointer metaphor (see Sec. 6.2.2) can be hard to accomplish. To this end, we replace the yaw and pitch angle values α_y^i and α_p^i of the current frame i with the angles of the last frame, which fulfilled the condition $|\alpha_r^i - \alpha_r^{i-1}| < 2(\left|\alpha_p^i - \alpha_p^{i-1}\right| + \left|\alpha_y^i - \alpha_y^{i-1}\right|)$. This way, the pointing direction remain constant during a clicking operation, because in this case the hand rotation is mainly around the roll-axis.

Note that exploiting one DOF for clicking purposes leaves us only 5 DOFs for other manipulations. Therefore, this technique can only be employed in specific interaction modes such as selection.

When this technique is currently applied in the interface, the visual feedback provides two small buttons (visualized as cylinders), positioned left and right alongside the hand model, which indicate the user how she/he can perform click-

ing. See Fig. 7.8 for an illustration.

Figure 7.8: The visual feedback for clicking simulation. Two additional buttons are rendered, one on the left and one on the right side of the hand model. Left: no button is pressed. Right: the virtual left button is pressed.

Furthermore, if currently the virtual pointer metaphor is used (e.g. for a point and select operation) the visual feedback is additionally modified as illustrated in Fig. 7.9. A ray is drawn illustrated as a line emanating from the hand model

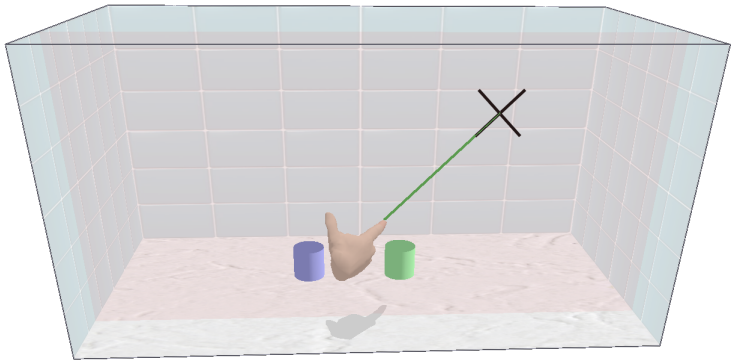


Figure 7.9: The visual feedback for virtual pointing. An additional green line originating from the hand model and a cross indicating the intersection point with the back plane is drawn.

toward the pointing direction. Additionally, the intersection point with the back plane is illustrated by a cross to indicate how the virtual cursor on the screen is moved.

In an informal user study this clicking technique could instantly be handled by everyone. Due to the proposed pointing direction modifications, the clicking operation itself did not reduce the precision of the selection task, even if the virtual pointer metaphor was used. Note that these modifications are only applicable, because the roll-axis angle is used for triggering the button events. The specific visual feedback of this technique supports mainly the familiarization with this technique. As this technique is very easy to handle, the support is only tem-

porarily. Nevertheless, it often helps as an indication which mode of interaction is currently applied.

7.4 Hybrid Cursor Control

The selection of virtual items or objects is typically solved by using a controller device to move a cursor above an item/object and issuing a selection command (e.g. using a button click). A controller device delivers one or more parameters (e.g. position, deflection) at discrete reading points (e.g. every 10 milliseconds a new position is delivered). In the context of 6 DOF hand-tracking, the hand acts as the controller device itself and a reading point is defined by the time the hand-tracking procedure succeeded. The movement of a 3D cursor is normally controlled by the position of one hand using the virtual hand metaphor (see Sec. 6.2.1) and the movement of a 2D cursor by the pointing direction using the virtual pointer metaphor (see Sec. 6.2.2).

The way of positioning a cursor can be expressed by a mapping from the parameters of the input device (e.g. 2D mouse, bare hand, etc.) to the coordinates of a cursor in the visualization space. We split this mapping up into a mapping g, which defines a mapping from the input device parameters to controller coordinates in the visualization space, and a mapping f, which defines a mapping from these controller coordinates to the final cursor coordinates in the same space:

$$\text{Device Parameters} \xrightarrow{g} \text{Controller Coordinates} \xrightarrow{f} \text{Cursor Coordinates}, \tag{7.4}$$

where

$$g: \quad \text{Device Parameters} \quad \rightarrow D_{coords}, \tag{7.5}$$

$$f: \quad D_{coords} \times \ldots \quad \rightarrow C_{coords}. \tag{7.6}$$

Thereby, $D_{coords} \subset R^d$ denotes the set of the controller coordinates, which is bounded by the effective range of the controller device. $C_{coords} \subset R^d$ denotes the set of the cursor coordinates bounded by the display boundary. d denotes the dimension of the visualization space (normally $d = 2$ or $d = 3$). This is illustrated in Fig. 7.10 for the 2D case. The mapping g determines how a device is used (e.g. hand position or hand pointing direction determines controller coordinates). f defines the cursor control and therefore maps from the controller coordinates to the cursor coordinates. f can additionally be dependent on other parameters like e.g. the controller velocity or the current cursor position. Using this formulation of two mappings we can redefine the way the cursor is controlled independently of the device. Therefore, we assume g to be given and focus on the mapping f.

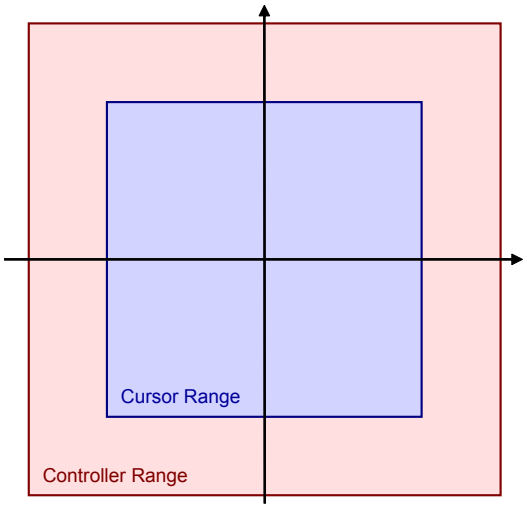


Figure 7.10: Exemplary 2D illustration of the range of the controller coordinates D_{coords} and the range of the cursor coordinates C_{coords} in the visualization space. The cursor coordinates are typically bounded by the display size and the controller coordinates by the effective range of the controller device. In this example, the center positions of the cursor and controller ranges coincide with the origin of the visualization space, which is not generally the case. Moreover, the sizes and size relations of the two ranges does not have a particular meaning.

If a controller device delivers absolute parameters (e.g. a hand-tracking device), also absolute positions in the controller range can be deduced at each reading point of the controller device. In this case, the current controller position p' (delivered at the last reading point) and the previous controller position p (delivered at the previous reading point) are explicitly given. Such devices will be referred to as absolute controller devices. However, other devices (e.g. the standard 2D mouse) do not provide absolute controller positions but only incremental information about the controller movement. In this case, only the vector of the controller position increment $p' - p$ is given and can be used to deduce the absolute cursor coordinates. Note that in return, by simply taking the difference of absolute controller positions, every absolute device can be used as an incremental device.

If f is described by an affine mapping with $f(p') = Ap' + b$ from the current controller position p' to the new cursor position c' with $c' = f(p')$, f describes an absolute cursor control. Thereby, A is a $d \times d$-matrix, b a vector of dimension d and $f : D_{coords} \rightarrow C_{coords}$. Note that f maps the absolute controller coordinates proportionally to the absolute cursor coordinates. Therefore, we call this kind of control an absolute proportional cursor control.

If an incremental device is used, the new absolute cursor position c' can be computed as a function f from the controller position increment $p' - p$ and the current absolute cursor position c. It can therefore be interpreted as a mapping $f : D_{coords} \times C_{coords} \rightarrow C_{coords}$, where $f(p' - p, c) = A(p' - p) + c$. A is again a linear mapping described by a $d \times d$-matrix. We call this kind of control an incremental proportional cursor control.

While both types of proportional cursor control are in general very intuitive, unfortunately, they either lack accuracy (if A scales up) or rapidness (if A scales down). Especially, if the addressable screen units are small (e.g. high resolution display) with respect to the range and resolution of the controller device, the selection of small virtual buttons or objects becomes unfeasible due to the low precision.

Therefore, the mapping f is often made dependent on further parameters such as the controller velocity. This way, an adaptive cursor control can be established. A prominent example is using a speed dependent cursor acceleration, where dependent on the current controller speed s' the cursor movement is accelerated or decelerated. For the incremental case, this looks as follows:

$$f(p' - p, c, s') = c + h(s')(p' - p), \tag{7.7}$$

where h denotes a function that is manipulating cursor acceleration (e.g. $h(s') = \text{diag}(s', \ldots, s')$). This function is often referred to as the display/control ratio, because it defines the ratio between the controller movements and the final cursor

movements on the display. This way, a non-linear mapping of real to virtual movements is applied which can facilitate both precise and fast cursor movements. The accuracy does not have to depend on the ratio between real and virtual working volume sizes/resolutions. However, an inherent problem of the adaptive cursor control is that during a cursor movement the controller might reach the boundary of the controller range although the cursor is not located at the boundary of the cursor range, yet. Therefore, to allow for adaptive cursor movements in the entire cursor range usually some kind of clutching mechanism is employed (i.e. a certain action couples/de-couples the virtual to/from real movements). For example in a normal desktop setup, cursor movements can be decoupled by lifting the mouse. This way the cursor can be moved step by step to every position in the cursor range independent of the bounded controller coordinates. However, using a clutching mechanism together with an absolute cursor control is not straightforward. Therefore, often also absolute devices are used in an incremental way and a clutching mechanism is employed to enable an adaptive cursor control.

While such a clutching mechanism is intuitive and easy-to-use for physical devices such as desktop mouses, the missing force feedback and lack of physical buttons lead to severe problems in developing and applying a suitable clutching mechanism for bare-handed interaction. Using different postures to specify whether the real and virtual movements are coupled [VB05] turned out to be too demanding for the fine motor skills for many people. Thresholding on the hand pose (e.g. Z-coordinate above a certain value) is not a good solution because force feedback is not available and therefore judging the clutching point is difficult. Therefore, recent approaches use either a two-handed control, explicitly switching between different positioning modes or clutching by thresholding schemes on the velocities and/or accelerations of hand movements. However, the inherent complexity and additional need of coordination degrade user acceptance and efficiency.

Also other free-hand devices are not suited to be used with a clutching mechanism due to interference with intuitiveness and user acceptance. For example the Nintendo Wii-Remote is used as a pointing device corresponding to pointing in real life. If instead a button on the controller is used for clutching most of the naturalness of the interaction is lost.

To overcome the need of clutching or mode switching for cursor positioning, we introduce a novel technique. Our technique is a mixture of absolute proportional and adaptive incremental cursor control and is related to PRISM [FKK07]. It does not need any introductions or assistance before or during interaction. Nevertheless, both precise and fast cursor movements can intuitively be performed.

7.4.1 PRISM

The most related pointing facilitation technique is *PRISM* (Precise and Rapid Interaction through Scaled Manipulation) [FKK07]. PRISM enables precise as well as fast movements by thresholding on the controller velocity combined with an offset recovery procedure. PRISM dynamically adjusts the D/C (display/control) ratio, which determines the mapping of controller to cursor movements and is in our formulation expressed by the function h. Note that a constant D/C ratio describes a proportional control. To determine the D/C ratio, PRISM uses three constant velocity thresholds: the minimum velocity ($MinS$), the Scaling Constant (SC) and the maximum velocity ($MaxS$). Then, the new cursor position c' is determined by using the following function h_{PRISM} as h in Eq. 7.7:

$$h_{PRISM}(s') = \begin{cases} 1 & \text{for} \quad s' \geq SC \\ \frac{s'}{SC} & \text{for} \quad MinS < s' < SC \\ 0 & \text{for} \quad s' \leq MinS \end{cases} \tag{7.8}$$

In PRISM, s' is defined as the controller velocity averaged over the last 500 milliseconds. If the current controller velocity is below $MinS$, any virtual movement is inhibited by setting the $h_{PRISM}(s')$ to zero. If the velocity is between $MinS$ and SC controller movement is quadratically mapped to cursor movement by linearly scaling with the current velocity. This way, slow controller movements are scaled down to gain a higher precision.

Simultaneously, an offset is accumulated that represents the cursor/controller displacement (i.e. the displacement between the controller and the cursor position). If the velocity is between SC and $MaxS$ the controller movement is constantly and directly mapped to cursor movement by setting $h(s')$ to one. In this case the offset does not change. If the velocity is above $MaxS$, additionally the accumulated offset is decreased by accelerating or decelerating the cursor to reduce the cursor/controller displacement. Independent on the current velocity, cursor movement is inhibited if the controller movement is directed toward the cursor position, as long as the offset is positive. Note that this way the offset is decreased and some kind of clutching mechanism is realized. Details can be found in [FKK07].

Originally, PRISM was designed to aid in manipulation of virtual objects directly with the user hand. In this case, not only a cursor is visualized but additionally a hand model, which facilitates the understanding of the relation between real and virtual movement. In the case of selection only a cursor is visualized. This complicates the understanding of PRISM. Especially, the de-clutching mechanism can lead to user irritations. For example, it often happens that the controller is moved to the boundary of its effective range (e.g. boundary of the working volume), but the cursor is still not located at the boundary of the cursor range.

This can lead to confusion about whether the user did something wrong or the controller device is not working correctly.

Because of the mentioned problems of PRISM for cursor control, a detailed introduction and quite long familiarization phase is needed to enable users suitably handling the device. This prohibits the employment of this technique for application scenarios not having this opportunity (e.g. for visitors on an exhibition). Furthermore, such difficulties decrease the general user acceptance.

7.4.2 Our Method

The basic idea of our technique is additionally making the function h explicitly dependent on the relative positions of cursor and controller. To allow for reasonably using the relative positions of cursor and controller, their coordinates have to be in a suitable relation. Furthermore, as we want to use an adaptive absolute control but avoid any clutching mechanism, we have to ensure that the cursor is always located at the boundary of the cursor range when the controller is at the limit of its effective range. Note that in general the cursor/controller displacement is changed when the cursor moves with a different velocity than the controller, for example to increase precision by slowing down the cursor. To this end, our technique has the following main features:

1. With increasing cursor/controller displacement the cursor is increasingly accelerated such that the entire cursor range can be reached without any special maneuvers.

2. With decreasing cursor/controller displacement the cursor is decreasingly decelerated such that the displacement is reduced.

3. Both precision and rapidness can be increased by an adaptive mapping.

4. No clutching mechanism or mode switching is used or needed.

To simplify the formulation of our method we define the cursor coordinates to range in $[-0.5; 0.5]^d$ (higher or lower values are clamped) and the controller coordinates in $[-1.5; 1.5]^d$ in the visualization space. This is illustrated in Fig. 7.11. Note that the controller range in the common coordinate system need to be larger than the cursor range to allow for reaching every position in the cursor range. Under these conditions we ensure that the cursor/controller displacement does never exceed 1. For simplicity, it is assumed that the range of the display pixel coordinates and the cursor range coincide. However, our technique can also be formulated more generally by a simple affine mapping between the display coordinates and the cursor coordinates.

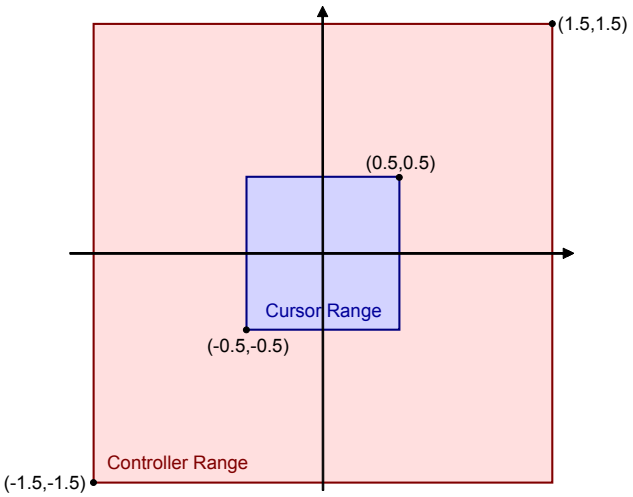


Figure 7.11: 2D illustration of the controller and cursor coordinates in the visualization space used for the current realization of our technique. Here, the sizes of the two ranges are normalized to the depicted extents and the centers of both ranges coincide with the origin.

The mapping f from the controller coordinates to the cursor coordinates for our technique looks as follows:

$$f(p, p', c, s') = c + h_{Hybrid}(p, p', c, s')(p' - p). \tag{7.9}$$

Note that h_{Hybrid} and f are now additionally dependent on the previous controller position p, the current controller position p' and the previous cursor position c.

The function h_{Hybrid} accelerates or decelerates the cursor movement depending on a combination of different cursor/controller distances. In our current realization, the following two cursor/controller distances are used: the l^2-norm of the cursor/controller displacement vector $v = \frac{1}{2}(p + p') - c$ with $d_2 = \|v\|$ and a signed distance d_1 that also depends on the movement direction. d_1 is defined as the length of the projection of v on the direction of controller movement with

$$d_1 = \left\langle v, \frac{p' - p}{\|p' - p\|} \right\rangle \tag{7.10}$$

and is illustrated in Fig. 7.12.

The sign of d_1 distinguishes whether the controller moves to ($d_1 < 0$) or from ($d_1 > 0$) the cursor position. Either a deceleration (if $d_1 < 0$) or an acceleration (if $d_1 > 0$) can reduce the cursor/controller displacement or the increase of cursor/controller displacement. The value of d_1 describes the size of the cursor/controller displacement dependent on the controller movement direction. We

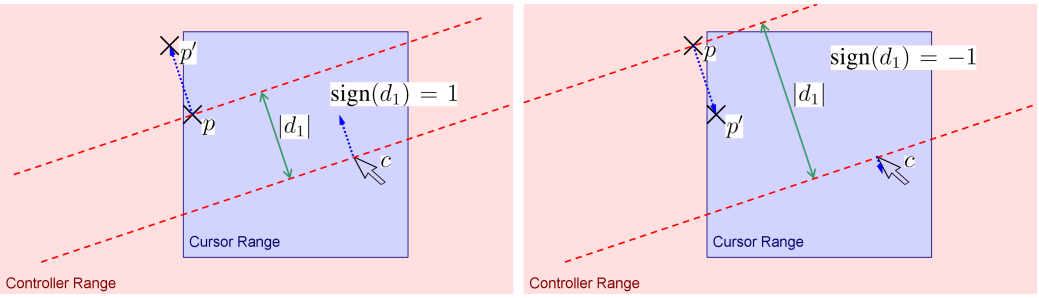


Figure 7.12: Illustration of d_1 in 2D. Left: Controller movement increasing the distance between controller and cursor position. Right: Controller movement decreasing the distance between controller and cursor position.

use it for indicating the amount of cursor acceleration/deceleration. Following this, we define h_{Hybrid} as

$$h_{Hybrid}(p, p', c, s') = \begin{cases} h(s') \cdot (1 - |d_1|) & \textbf{for} \quad d_1 < 0 \\ h(s') + (1 - d_2)d_1 + d_2 d_2 & \textbf{for} \quad d_1 \geq 0 \end{cases} . \quad (7.11)$$

In the first case the cursor movement is decelerated depending on the value of d_1. Note that in this case $|d_1|$ never becomes greater 1. In the second case the cursor movement is accelerated depending on the values of d_1 and d_2. d_2 is used to weight the influence of d_1 and itself. This weighting is needed to ensure $|d_1|, d_2 \leq 1$, because otherwise (if simply $h(s') + d_1$ would be used) the displacement could become larger than 1 if the controller is moved orthogonal to the displacement vector (if $d_1 = 0$). In our current realization, the speed dependent acceleration is defined as in PRISM with $h(s') = h_{PRISM}(s')$ (see Eq. 7.8), but $MinS$ is constantly set to zero in order to prohibit moving the controller over a great distance without cursor movement.

However, one problem of our method appears if the user moves the cursor slowly over a large distance in order to perform a precise operation at the reached position. In this case, the precision is reduced due to the large displacement between cursor and controller. Therefore, our technique is not suited for application scenarios in which such tasks have to be carried out very often. However, in menu navigation and selection tasks (whether 2D or 3D) typically users first perform a fast movement to coarsely reach a desired position and then perform slow movements for precise operations. In these cases, our method leads to superior results. Moreover, no introduction or practicing is needed for an effective handling.

For an illustration of 2D cursor positioning using absolute proportional control, PRISM and our technique see the first accompanying video for bare-handed interaction. In the second accompanying video cursor positioning is illustrated for

the Nintendo Wii-Remote controller using our technique.

7.4.3 Experiments

We conducted a user experiment to evaluate the performance and usability of the Hybrid Cursor Control technique. Therefore, an objective testing scenario as well as a user questionnaire were part of the study. In the objective test, our hypothesis was that our technique would be superior to other state-of-the-art cursor positioning techniques in terms of efficiency for menu navigation and selection tasks especially if both precision and rapidness are required. In order to investigate this hypothesis, we compared the Hybrid Cursor Control (HCC) to absolute proportional cursor control (PC) as well as PRISM in a button clicking scenario.

Regarding the questionnaires we expected that our technique would be rated superior to PC as well as to PRISM in terms of comfort, ease of use and liking/preference. Furthermore, we anticipated our technique to be rated more accurate than PC and more intuitive than PRISM.

Experimental Setting

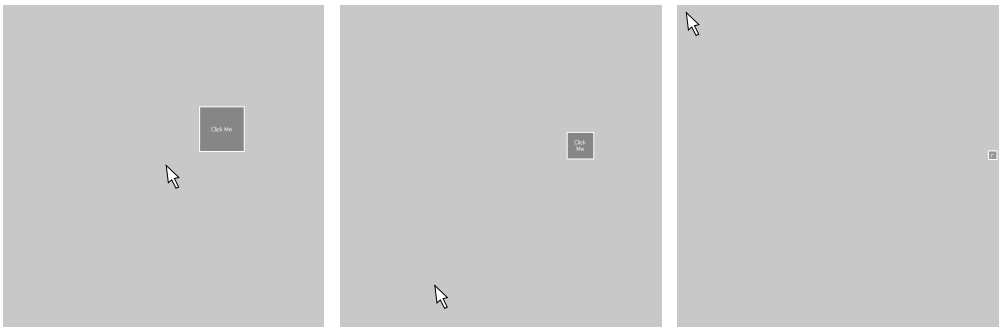
Two connected PC-based systems were used in the experiment, one coupled to the cameras for tracking the hand (see Sec. 2.5) and another (Intel Core 2 Quad Q8200, Geforce 8800) for running the test application. The application was visualized on a standard 19" TFT-Display.

Experimental Task

In the experimental task 2D buttons of different sizes being located at different distances from the 2D mouse cursor have to be clicked by positioning the cursor above a button and performing a left click Roll Click (see Sec.7.3). For cursor positioning the virtual pointer metaphor (see Sec. 6.2.2) combined with the respective positioning technique (either PC, PRISM or HCC) is employed. To obtain a level of difficulty the button size is successively decreased while the distance from the cursor to the button is successively increased both in 10 equally spaced steps. Therefore, after each button click the button is repositioned in a random direction as illustrated in Fig. 7.13.

For each button click three different values are measured:

1. The time needed to roughly approach the button (equal for all button sizes), which is achieved when the distance from the cursor to the button center becomes less than a certain threshold. This time reflects the rapidness of the respective cursor positioning technique.



(a) Step 1 (b) Step 5 (c) Step 9

Figure 7.13: Three steps of the task that had to be solved. The button size is successively decreased and the distance to the cursor is increased (from left to right).

2. The time beginning when the button is roughly approached and ending when it is finally clicked. This time reflects the preciseness of the cursor positioning technique.

3. The number of click errors. Every click that is performed while the cursor is not located above the button is considered as a click error. Note that the number of click errors is equal to the number of clicks minus one.

Questionnaires

After the practical tasks, each subject was asked to complete a short questionnaire. This questionnaire consists of six ratings in terms of ease of use, control accuracy, intuitiveness, learnability, comfort and liking/preference for each cursor positioning technique. Thereby, the typical 5-point likert rating scales [Lik32] were used, where the subjects had to choose a number from 1 to 5 using the following criteria: 1 - very bad, 2 - bad, 3 - neutral, 4 - good, 5 - very good. Moreover, the questionnaire comprised a section for additional comments and a section for personal details such as age, gender, dominant hand and experience with free-hand pointing.

Participants

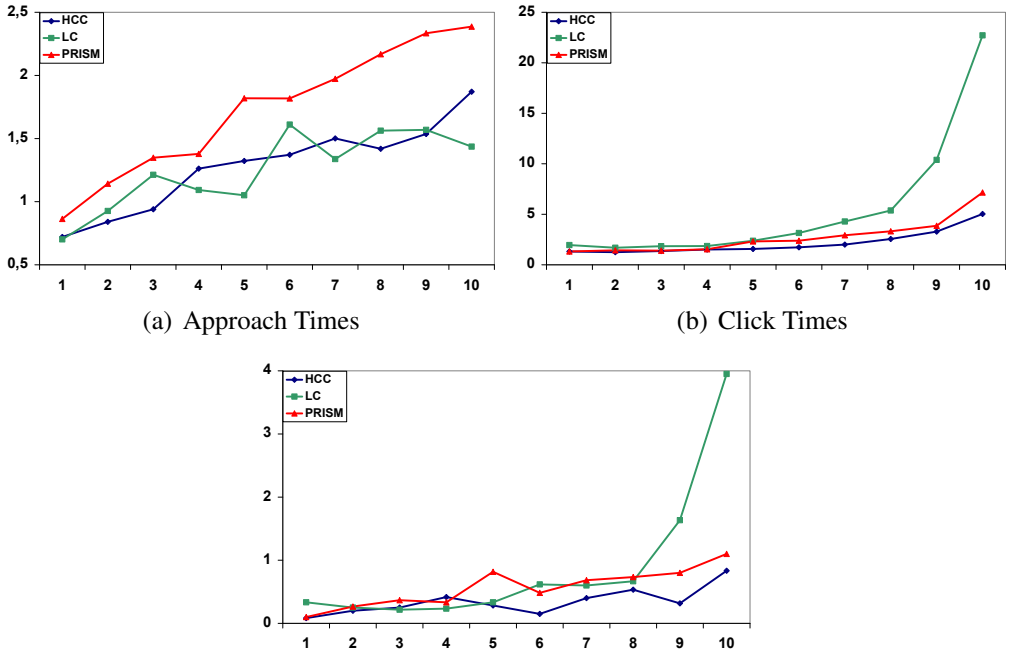
Twelve participants (ten male, two female) in the age of 21 to 34 with an average age of 28 took part in the experiment. Only one person is left handed. Most had little or no experience with bare-handed interaction. Three participants were already experienced in free-hand-pointing using the Nintendo Wii-Remote controller.

Procedure

Each participant had to solve the task (one task means 10 times clicking a button of decreasing size) five times by employing each of the three positioning techniques. Thereby, the sequence of the employed techniques was permuted evenly and all tasks had to be finished before the next technique was adopted. In advance to the testing of each technique its mode of action was explained and the subjects could familiarize with it in a short preparation time (up to two minutes).

Results and Discussion

The average approach times, the average precision times and the average click errors are illustrated in Fig. 7.14.



(a) Approach Times

(b) Click Times

(c) Number of Errors

Figure 7.14: Mean completion times (in seconds) and mean quantity of click errors each for 10 different levels of difficulty (x-axis). Note that the y-axis scale of the click times is significantly larger than of the approach times.

For roughly approaching the target the absolute proportional control (PC) shows a better performance than PRISM and a similar performance as the Hybrid Cursor Control (HCC), which is due to the rapidness of absolute positioning. Moreover, the click times and error rates of PC are similar good as of PRISM and

HCC for large buttons (low level of difficulty). However, PC turned out to be inappropriate for clicking small buttons because already slight natural hand tremor and hand-tracking inaccuracies substantially affect the cursor precision. This is reflected in the click timings and error rates for higher levels of difficulty, where the values of PC are up to four times higher than the respective values of HCC and PRISM.

PRISM has consistently higher timings than PC and HCC for approaching the target, but as these differences are less than one second in average and the approach times generally are lesser than the click times this affects the overall performance only marginally. Furthermore, in contrast to PC also high levels of difficulty could suitably be solved using PRISM, the according click times and error rates are considerably lower. In lower levels of difficulty the results of PC and PRISM were similar.

Using HCC also all levels of difficulty could suitably be solved. Moreover, nearly for all measures (approach times, click times and errors) and all levels of difficulty HCC shows a better average performance than PRISM. The performance gain was even larger in higher levels of difficulty, because most subjects had difficulties to suitably handle the implicit clutching mechanism of PRISM. Compared to PC, HCC certainly performed considerably better in high levels of difficulty, but also in lower levels of difficulty the results were superior.

The results obtained by the questionnaires are illustrated in Fig. 7.15. Note

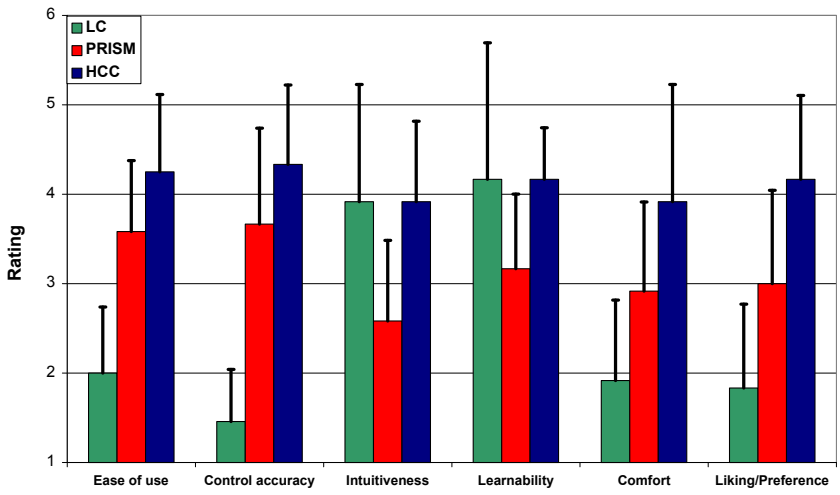


Figure 7.15: Illustration of the mean ratings for the different categories and cursor positioning techniques that were collected by the questionnaires. The error bars indicate the respective standard deviations.

that the high standard deviations are primarily due to the individual affinity for either high or low ratings. However, the rankings of the different techniques were

mostly the same. For example several subjects rated PC, PRISM and HCC with 2, 3 and 4 in a category, while other subjects rated them with 3, 4 and 5 in the same category. Nevertheless, some statistical significances could be derived by performing the Student's t-test, where a confidence equal to or greater than 95% is considered to be significant.

The results of intuitiveness and learnability are similar. Both PC and HCC were rated significantly superior to PRISM, which is due to the more complex handling of PRISM. The average rates of PC and HCC are nearly equal in both categories, but the standard deviations of PC are considerably higher than of HCC, which means that the subjects' ratings for PC are more inconsistent. This is because some subjects found PC to be more intuitive than both other techniques as it directly corresponds to pointing in real life, while in contrast some other subjects found it to be less intuitive due to the missing cursor acceleration. They argued that cursor acceleration is intuitive as it is usual in the employment of standard 2D mouses.

In the other categories ease of use, control accuracy, comfort and liking the rankings of the different techniques are equal, HCC was rated best, followed by PRISM. In all of these categories PC is rated significantly inferior to HCC as well as to PRISM, which is due to PC's inapplicability for suitably clicking small items. In terms of comfort and liking also PRISM was rated significantly inferior to HCC. It was constituted by most of the subjects that they did not like the clutching mechanism which makes PRISM more unintuitive. In terms of ease of use and control accuracy PRISM was rated also inferior to HCC, but the differences are not as significant, because clicking small buttons was also suitable using PRISM.

Regarding both the performance testing results and the results from the questionnaires our novel technique seems to be a promising alternative for free-hand pointing facilitation. Existing problems such as cursor precision and unintuitive handling are solved. This way, menu navigation and selection with smaller target sizes become possible and the user acceptance is increased.

7.5 Virtual Camera Steering

In a 3D virtual environment one fundamental operation is moving the virtual observer/camera. To this end, we decided to use the traditional and commonly used approach of steering, which was introduced by Ware and Osborne [WO90] and named the flying vehicle control metaphor. The translational and rotational distances of the user's hand pose from a certain resting pose determine the translational and rotational velocities of the virtual camera to the according directions.

We use the 'pointingA' posture (see Sec. 3) to steer the virtual camera. The squared deviation from the resting pose is used to control the respective speed. For

example, while the index finger points toward the screen and the thumb upwards, a small turn of the thumb toward the left increases the rotational speed of the virtual camera toward the left, and moving it up again stops the rotation. Analogously, the viewing direction can be controlled intuitively by changing the direction of the index finger. In a similar way moving to the left or right, front or back and up or down is performed by translating the hand from its resting position to the corresponding directions, respectively, and thus increasing the speed toward these directions.

The 'pointingB' posture (see Sec. 3) is used to stop the camera movement. Furthermore, when the posture has been changed back from the 'pointingB' to the 'pointingA' posture, the resting position is set to the position of the hand when reaching the 'pointingA' posture. We also experimented with resetting the entire pose (i.e. also the orientation) this way, but unfortunately, several subjects were not capable to handle it. The resting orientation was often unintentionally set to an unusable configuration which led to disorientation in the interaction. Instead resetting only the resting position could easily handled by all subjects.

Several subjects experimented with the fly through application using either our hand-tracking device or a standard 6 DOF space mouse (3D connexion SpaceNavigator). The supposed intuitiveness of using hand-tracking was fully confirmed. All test persons could control the 6 DOF with our hand-tracking device after a short familiarization with the great amount of freedom, whereby using the space mouse some failed completely. Especially to combine different DOF was significantly easier by using the hand-tracking device.

This control mode enables the user to quickly reach every position and orientation in the scene. However, if she/he loses sight of e.g. an object she/he currently intend to inspect, it is sometimes hardly possible to turn the camera back due to the lack of orientation. In the case of 3D object manipulation typically one or more objects are selected. Therefore, we assume that these objects are in the focus of the user and provide an additional visual feedback. We superimpose 2D arrows pointing to the boundary of the viewport indicating how to turn the camera back to the objects if they got out of sight. For example if the user turns or moves the camera toward the left such that the selected objects leave the window at the right side, an arrow is superimposed on the right side of the window, which is pointing toward the right. This way the user knows that if she/he turns the camera back toward the right the objects will get back into focus. Analogously arrows are superimposed at the left, top or bottom of the window if none of the selected objects is in focus and located in this direction.

If the camera steering is active, also several modifications are applied to the visual feedback as shown in Fig. 7.16. Due to the lack of haptic feedback users have often problems to return their hands to the resting pose in order to stop moving the camera. To this end, in our visual feedback the resting position is visualized

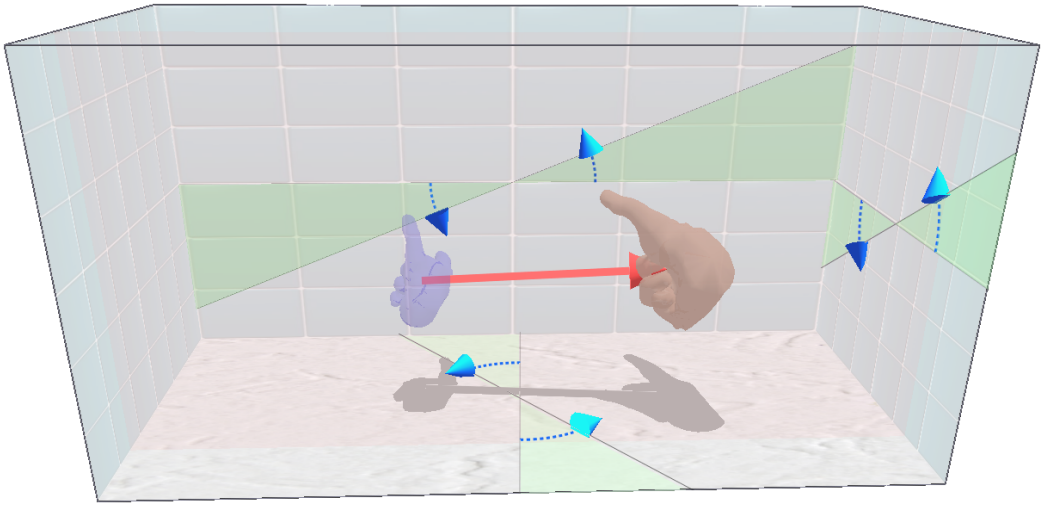


Figure 7.16: The visual feedback for moving the camera.

by showing a static additional hand model (blue, half-transparent). To further hint the currently applied translational camera movement, a 3D arrow (red) originating from this static hand model and ending at the currently moving hand model is provided. The orientation is split into roll, pitch and yaw angle and the resting orientation is illustrated by three static lines on the back wall, side wall, and floor of the cube, respectively. Simultaneously, three rotating lines indicate the currently applied rotation. The area between each of the rotating lines and its according static line is shaded (bright green) and two circular arrows (blue) are depicted, respectively.

In particular for this specific visual feedback we got a clear positive response from the subjects in our informal user study. We argue that due to the less intuitive handling of the steering technique the support the visual feedback can provide is higher.

7.6 Mode Switching

In order to be able to switch between different manipulation tools or adjust application settings we chose a similar approach as used in standard interfaces, where the user can switch between a menu and manipulation mode. In the menu mode, the different interaction modes/settings can be selected/changed from a 3D toolbar, and in the manipulation mode the selected interaction mode is applied. To switch between the two modes, the user can choose between two different techniques, described in the following subsections.

7.6.1 Working Volume Split

The 3D working volume is divided into a near region and a far region as depicted in Fig. 7.17(Left). If the user's hand is located in the near region, the menu mode

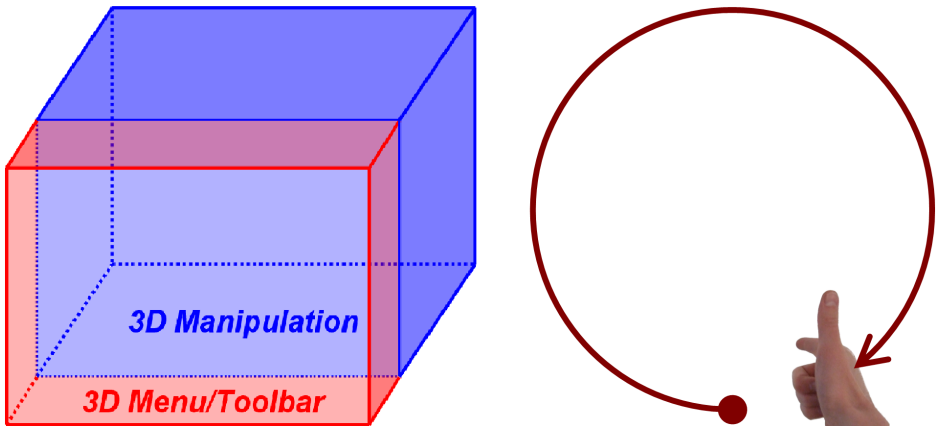


Figure 7.17: Left: Partitioning of the working volume. The z-coordinate of the hand position determines menu or manipulation mode. Right: A circular free-hand gesture performed jerky and fast switches between menu and manipulation mode.

is chosen. When the hand enters the far region, the selected interaction mode is applied. Thereby the far region is about four times larger than the near region in order to gain enough space for manipulation.

This way, switching between menu and manipulation mode is simple, but the available manipulation space is reduced and some distraction results from unintentional menu/manipulation transitions.

7.6.2 Free-Hand Gestures

The user switches between menu and manipulation mode by performing a certain free-hand gesture (we used a circle in the xy-plane, see Fig. 7.17(Right)), while her/his hand remains in the 'Not attached' state (see Sec. 7.2). This way, the entire working volume can be exploited for manipulation purposes, but switching between menu and manipulation is more difficult. As both solutions have their advantages and disadvantages, the user can choose between them. If for example a single long manipulation step is planned, she/he could select the second alternative and otherwise the first one.

Due to the fact that fast and jerky movements are primarily suitable for coarse operations, exploiting hand movements in the 'Not attached' state for recognizing free-hand gestures turned out to be reasonable. Therefore, we additionally enable

free-hand gestures for switching between interaction modes directly. For example, by performing a gesture corresponding to the letter S the user can directly switch between object manipulation and object selection. This way, a sequence of several selection and manipulation cycles can be solved more efficiently.

CHAPTER 8

TWO-HANDED TECHNIQUES

In the previous chapter several techniques for one-handed interaction were presented. As natural interaction with hands typically involves both user hands, we also investigated how the second hand can be used for improving interaction. In the context of bi-manual symmetric manipulation it is known that two hands can increase precision and intuition (see e.g. [OKF+05]). But still the problem of suitably coupling and decoupling a virtual object's movements to the user's hands (i.e. grabbing and releasing) has to be solved. Using the Jerky Release technique (see Sec. 7) is one possibility, but in the case of two-handed interaction we could establish a more natural as well as more efficient solution. Additionally, we solved the inherent problem of intuitively mapping the poses of both hands to the pose of one virtual object. The resulting technique is introduced in the first section of this chapter. In the second section, we introduce several simple solutions for compensating some general drawbacks that appear if vision based tracking is employed for two hands simultaneously. Last but not least a short evaluation and discussion is as well presented, which is based on a user study, user questioning and our observations.

8.1 Bi-manual Symmetric Grab

For our grabbing and releasing technique we assume an object has been selected and now the user's intention is manipulating the object's pose by exploiting both hands in order to gain a high precision and control. In this case an additional suitable mechanism is needed for determining when the objects movements shall be coupled to the hands' movements or not (grabbed or released). This mechanism should enable precise releasing of objects and should be manageable efficiently and intuitively. To this end, we introduce an approach based on different velocities of the grabbing/releasing action. The idea is to trigger a grabbing action if the hands move together while a release action is performed if the hands move apart as illustrated in Fig. 8.1.

Grabbing and releasing of an object can be expressed by a state machine comprising the two states 'Grabbed' and 'Released' and the transitions in between. A

Figure 8.1: Illustration of real two-handed grabbing, manipulating and releasing. The red arrows indicate either a grabbing gesture (Left) or a releasing gesture (Right).

reasonable formulation of the exact conditions is crucial in order to avoid involuntary grabbing or releasing actions. The user's actions have to be suitably analyzed to faithfully distinguish grabbing, manipulation and releasing. To this end, the conditions are based on the grabbing velocity v_G^i and the manipulation velocity v_M^i. v_G^i denotes the signed velocity with which the hands moved together (positive) or apart (negative) between the previous frame $i-1$ and the current frame i and is defined as:

$$v_G^i := \frac{\|p_L^{i-1} - p_R^{i-1}\| - \|p_L^i - p_R^i\|}{t^i - t^{i-1}}, \tag{8.1}$$

where p_L^i, p_L^{i-1}, p_R^i and p_R^{i-1} are the tracked positions of the left and right hand in frame i and $i-1$, respectively. t^i and t^{i-1} denote the times of frame i and $i-1$. The manipulation velocity v_M^i denotes the sum of the hands' translational velocities that will potentially be used for object manipulation. In other words v_M^i is the sum of the translational velocities of both hands minus the grabbing velocity and is defined as:

$$v_M^i := \frac{\|p_L^i - p_L^{i-1}\|}{t^i - t^{i-1}} + \frac{\|p_R^i - p_R^{i-1}\|}{t^i - t^{i-1}} - |v_G^i|. \tag{8.2}$$

Now, in order to discover whether the user wants to perform a grab/release action or simply an object manipulation, the grabbing and manipulation velocities are analyzed. To this end, we introduce the modified signed grabbing velocity $\tilde{v}_G^i$ which is defined as:

$$\tilde{v}_G^i := \begin{cases} \textbf{sgn}\,(v_G^i)\,(|v_G^i| - v_M^i) & , \; if \;\; |v_G^i| > v_M^i \\ 0 & , \; else \end{cases} \tag{8.3}$$

Thereby, **sgn** denotes the sign function. Note that $\tilde{v}_G^i$ is zero while the manipulation velocity is either dominant or equal to the grabbing velocity. If the grabbing

velocity is dominant $\tilde{v}_G^i$ is either positive (the hands perform a grab movement) or negative (the hands perform a release movement). Now, our grabbing and releasing technique can be expressed by the state machine depicted in Fig. 8.2. T_G

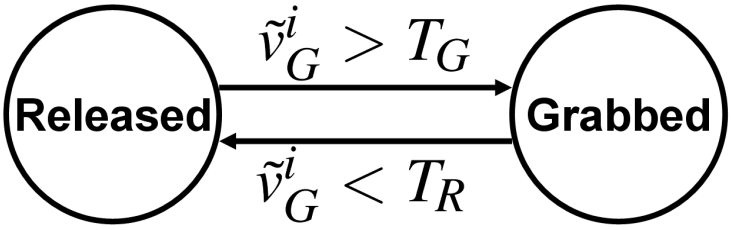


Figure 8.2: State machine illustrating how an object can be grabbed and released. Thresholds T_G and T_R are applied to the modified signed grabbing velocity $\tilde{v}_G^i$.

and T_R are thresholds for triggering a grab or release action, whereby T_G has to be a positive and T_R a negative real value. These parameters are used to adjust the velocity that has to be performed by the user to grab or release an object. In our current setting we use symmetric thresholds with $T_G = 10\frac{cm}{s}$ and $T_R = -10\frac{cm}{s}$. These values were determined in a short test scenario, where several users should perform this kind of grabbing and releasing gestures and manipulation movements while we recorded the employed velocities. We categorized the velocities manually to belong to the 'Grabbed' or 'Released' state. By analyzing several of these sequences we identified the threshold values, that would lead to the fewest errors. An error can either be a falsely triggered grab/release action or an intentional grab/release action that was not triggered.

Using this formulation the grabbing and releasing of a virtual object can be performed with only a slight movement; no spacious and therefore uncomfortable gestures are needed. Moreover, employing the manipulation velocity to inhibit grab/release actions avoids involuntary grabbing/releasing during a manipulation task. For the same reason the velocity thresholds T_G and T_R can be chosen with such a low value which enables grabbing/releasing with slow movements. Note that these thresholds have to be greater than zero because otherwise unintentional grab/release actions could occur due to the natural tremor of human hands or slight inaccuracies of the hand-tracking device.

Furthermore, this formulation allows for grabbing and holding an object with arbitrary distance between the hands. This supports the adaptation to different demands of the current task, e.g. if a very precise rotation shall be performed a greater distance between the hands is superior while a smaller distance is more convenient for large translational manipulations due to the limited work space. Note that a simple solution as for example using a specified distance between the hands to trigger grabbing and releasing would not have these features.

Using this criterion for releasing operations enables fairly precise positioning of objects. However, humans in general do not perform release operations with perfect symmetric movements of both hands and only in the grabbing direction. Hence, sometimes slight unintentional movements occur while the user wants to release the object but the velocity threshold T_R is still not exceeded. To overcome this problem, we inhibit object movements while $\tilde{v}_G^i < 0$ causing the object to stand still while the grabbing velocity is dominant and the hands perform very slow releasing movements. This way, the object's movements stop immediately when the user starts to perform a release action. Note that this condition generally imposes no restriction on intended object manipulations, because in these cases the manipulation velocity is dominant and $\tilde{v}_G^i$ is equal to zero. Applying this simple modification to our two-handed grabbing/releasing technique the precision for positioning virtual objects can further be improved.

Additionally, our technique enables performing grab and release actions in a comfortable and effective way. Furthermore, due to the close relation to natural two-handed grabbing it is intuitive and no long practicing is needed.

8.1.1 Object Movement Modification

To allow for bi-manually moving an object we developed a manipulation technique that is inspired by the grab-and-twirl technique introduced by Cutler et al. [CFH97]. The grab-and-twirl technique can be formalized as follows. The translational object movement is determined by the average translational movements of both hands. The rotational object movement is determined by two different perpendicular rotations: the rotation of the line connecting the two hand positions and the average hand rotations around this line. Both rotations are applied to the object. Note that Cutler et al. [CFH97] proposed to use not the average hand rotations but instead the rotation of only one hand. However, because we observed the combination of both enabling more precise rotations we preferred to use the average.

In contrast to the grab-and-twirl technique we do not trigger grabbing and releasing of an object by moving the hands' virtual representations until they both intersect the object and then performing a grabbing posture. Instead, using the technique introduced in Sec. 8.1 grabbing and releasing can take place independent of the size or position of the virtual object or the distance between the hands. Therefore, grabbing and releasing is significantly easier and more comfortable, but in order to preserve the intuitiveness of object movements the hands' physical movements must not be mapped to the virtual world in the same way as done in the grab-and-twirl technique. In the grab-and-twirl technique an object is translated by a certain mapping of the physical center between the two hands to the virtual center of the object. This is reasonable, because the virtual hand positions

coincide with the two pivot points where the object is grabbed (see Fig. 8.3(Top left)). In our technique, the user also anticipates two pivot points being located close to the object's surface where it is grabbed, but as in our case the distance between the hands' positions is not fixed to the object size, the same mapping as in Fig. 8.3(Top left) of physical to virtual movements would lead to unnatural results (see Fig. 8.3(Top right)). To obtain reasonable results the mapping has to be modified as illustrated in Fig. 8.3(Bottom left).

Unfortunately, if a solution which simply scales the translation by the ratio of the distance between the two pivot points (determined by the two intersection points of the object's convex hull with the line through the object center having the direction of the line connecting the two hand positions) to the distance between the hands would be applied, small objects could hardly be translated any more as illustrated in Fig. 8.3(Bottom right). Our solution consists in two steps: first, we adjust the hands' positions such that their distance is normalized and second, we split the translational movement up into an asymmetric part and a symmetric part which are scaled differently.

First, the left and right hand positions p_L^i and p_R^i in the current frame i are adjusted to have the same distance as in the previous frame $i-1$. The adjusted position for p_L^i is obtained by moving p_L^i in direction of p_R^i with a certain signed distance. This signed distance is equal to the difference between the hand distances d^i and d^{i-1} in frame i and $i-1$, respectively, multiplied by a suitable weighting factor λ_L^i. The scheme for obtaining the adjusted left hand position $\tilde{p}_L^i$ is defined as:

$$\tilde{p}_L^i := \lambda_L^i \ \ (d^i - d^{i-1}) \ \ \frac{p_R^i - p_L^i}{\|p_R^i - p_L^i\|} + p_L^i. \tag{8.4}$$

The weighting factor λ_L^i is defined to be the ratio of the translational amount of the right hand to the sum of translational amounts of both hands and can be formalized as follows:

$$\lambda_L^i = \frac{\|p_R^i - p_R^{i-1}\|}{\|p_R^i - p_R^{i-1}\| + \|p_L^i - p_L^{i-1}\|}. \tag{8.5}$$

$\tilde{p}_R^i$ is determined analogously by swapping all L's and R's in both equations. This way the hand that moved slower becomes stickier for the object (i.e. the object tends to stay in touch with this hand). If for example one hand remains at the same position the object can not be translated toward the grabbing direction by movements of the other hand. Furthermore, no more translational object movements are induced by decreasing or increasing the distance between the hands. Note that λ_L^i is equal to $(1 - \lambda_R^i)$ and $\lambda_L^i \in [0; 1]$, therefore the distance between the adjusted hand positions is guaranteed to be equal to the distance between the previous hand positions.

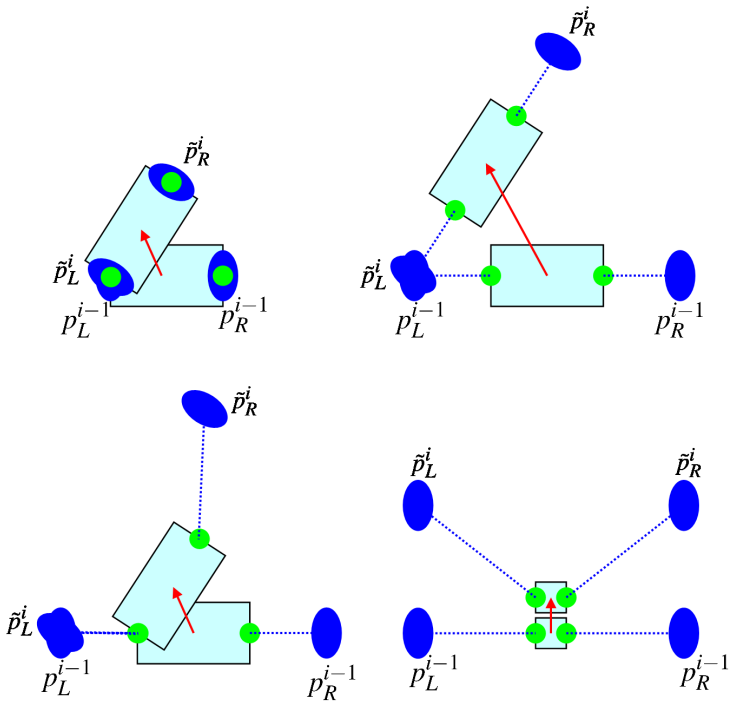


Figure 8.3: Illustration of the problem of mapping real to virtual coordinates with fixed distance between the hands. An object (rectangular box) is moved by exploiting the positions of both hands (blue ellipses). The pivot points are illustrated in green and the applied translation vector in red. Top left: distance between hands is equal to distance between pivot points. Top right: hand distance is greater than distance between pivot points but the same mapping is used as in (Top left). Bottom left: hand distance is greater than distance between pivot points and a suitable mapping is used. Bottom right: hand distance is much greater than distance between pivot points and a scaling by the ratio of pivot point distance to hand position distance is used.

Second, the translational movement is split up into an asymmetric part (difference of the hand translation amounts) and a symmetric part (dissimilarity of the hand translation amounts). The asymmetric part describes the amount of translation that influences the position of the center of rotation. This part has to be applied in a one-to-one mapping (i.e. it has to be mapped as in Fig. 8.3(Bottom left)). The symmetric part describes the amount of translation that has no influence on the displacement of the rotation center. This part can be applied to the object translation with an arbitrary scaling (independent of the object size or the hand distance) without affecting the intuitiveness. The two parts can be computed as follows: if t_L^i and t_R^i are the left and right hand translations with $t_{L/R}^i = \tilde{p}_{L/R}^i - p_{L/R}^{i-1}$ in frame i, then $t^i := \frac{1}{2}(t_L^i + t_R^i)$ describes the total amount of translation. We then define λ^i to be the percentage of symmetric translation with

$$\lambda^i = \frac{\min(\|t_L^i\|, \|t_R^i\|)}{\max(\|t_L^i\|, \|t_R^i\|)}. \tag{8.6}$$

Now, the resulting modified total translation $\tilde{t}^i$ can be written as:

$$\tilde{t}^i = s\lambda^i t^i + r(1-\lambda^i)t^i, \tag{8.7}$$

where r is defined to be the ratio of object extend to the current distance between the hands and s can be chosen arbitrarily to alter the mapping from real to virtual coordinates. This way, if an asymmetric rotation is performed (e.g. one hand is moved fast, the other stands still) the object is rotated intuitively around the pivot point touching the object and corresponding to the hand that was not moved (see Fig. 8.4(Left)). On the contrary, if a symmetric movement of both hands in equal directions is performed the object translation does not depend on the object size or distance between the hands (see Fig. 8.4(Right)).

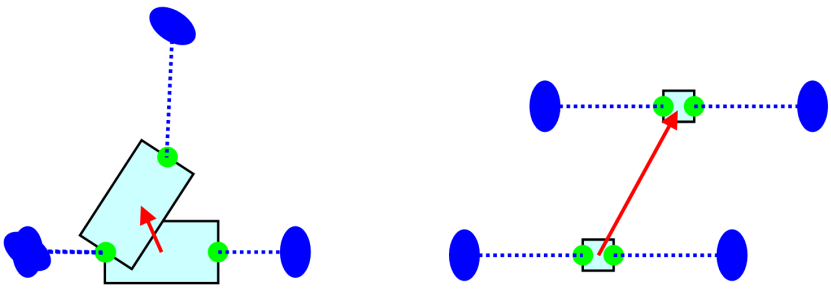


Figure 8.4: Illustration how our technique maps real to virtual movement. Left: Asymmetric movement. Right: Symmetric movement.

In order to provide a visual clue about where the object is grabbed and how it can be turned around, the resulting two pivot points are visualized in the 3D

interface. Two spheres are drawn at the corresponding positions as long as the object is grabbed. Additionally, this gives a direct visual feedback about whether the object is grabbed or released. An illustration is given in Fig. 8.5.

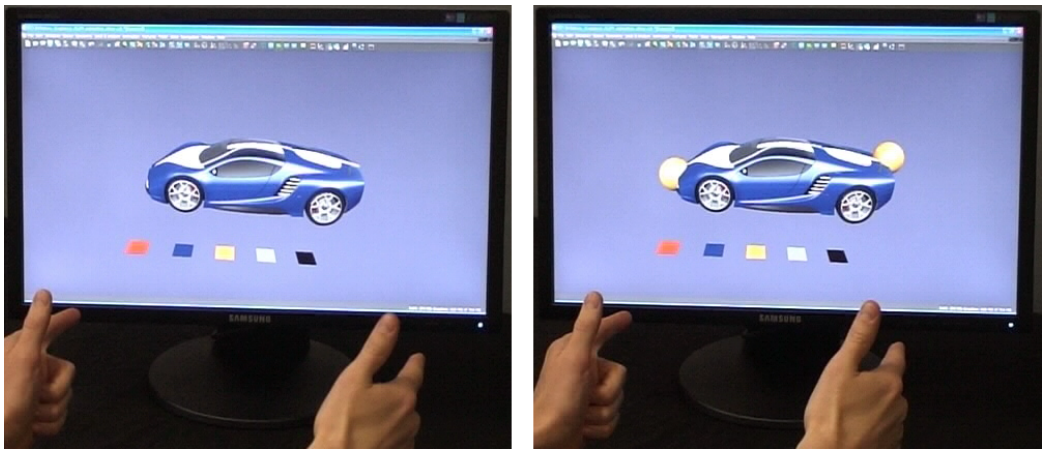


Figure 8.5: Visual feedback for grabbing and releasing. Left: The object is released. The pivot points are not visualized. Right: The object is grabbed. The two yellow spheres indicate the pivot points and that the object is grabbed. (Car model courtesy of RTT AG)

With this technique both human hands are exploited for simultaneous manipulation of the whole 6 DOFs of an object's pose in order to gain a significantly improved rotational precision and a generally higher control over the object. These advantages are due to the following reasons: first, the rotation depends on the distance between the hands leading to an increased precision if the distance is increased. Second, the center of rotation can be chosen more intuitively than in one-handed interaction, because object rotations are determined by hand translations (e.g. fixing one hand to a certain position leads to rotating an object around this point when the other hand is moved).

8.1.2 Integration with One-Handed Techniques

We argue that symmetric two-handed object manipulation is primarily suitable as an extension of one-handed manipulation. Only if high precision and/or control of the object is needed the application of both hands can significantly improve the current task. A reasonable scenario could be as follows: a user selects an object and moves it to the approximate pose by employing only one hand. Only then she/he uses both hands for fine adjustment of the object pose. In order to allow

for such scenarios in an effective way, transitioning from single-handed to two-handed interaction and backwards must be manageably fast and uncomplicated.

The first case of switching from single to two-handed interaction can simply be solved by automatically enabling the two-handed interaction mode when the second hand enters the working volume.

The second case of changing from two to single-handed interaction is more complicated. This is due to the problem that a hand could have left the working volume unintentionally. This occurs if the user currently performs an object manipulation at the edge of the working volume and moves one hand outside by mistake. To this end, we distinguish between two cases: first, if the object was grabbed when the hand left the working volume we assume leaving was unintended and prompt a message on the screen to move the hand back inside. Second, if the object was already released we assume the user wants to switch back to the one-handed manipulation mode which is therefore automatically enabled in this case. This distinction turned out to be intuitive and easy to handle.

8.1.3 Stabilizing the Object Movement

Exploiting the human hands as direct input devices by using vision based markerless hand-tracking has several great advantages (e.g. intuitive, non-obtrusive, etc.). However, as well some problems arise when such devices are exploited. In the following we explain some major drawbacks and present our solutions.

One drawback of vision based markerless hand-tracking is the dependency on the hand segmentation in the camera images. Performing a good segmentation becomes even harder when two hands are tracked simultaneously due to reciprocal occlusions. Such occlusions occur more often if the distance between the hands decreases. We observed the tracking stability of the hands' orientations to be primarily sensitive to such hand configurations. Note that in the employed two-handed object manipulation technique the hand orientations are only exploited for rotating the object around the line connecting the two hand positions (see Sec. 8.1.1). In order to avoid distracting rotational jumps during object manipulation we propose the following simple stabilization procedure: if the angular velocity of one hand exceeds a certain threshold (currently we use $2\pi\frac{rad}{s}$) its rotation is set to the identity. This threshold value was determined by analyzing the typical velocities of rotational jumps induced by an incorrect tracking. This way, the jumps of the hands' orientations are not applied in the current step leading to a stable object manipulation. In return, very fast rotations around this axis can not be performed any more. This is a clear limitation, but as it counts only for velocities higher than $2\pi\frac{rad}{s}$ the interaction is marginally affected. In practice, we discovered such high angular velocities to be exploited for less than 1% of object rotations in typical manipulation tasks.

Another problem induced by segmentation issues is that tracking two hands is impossible when they can not be separated in each camera image. In this case, a message is prompted requesting the user to increase the distance between her/his hands.

In particular relevant for two-handed object manipulations is the problem of limited work space; the hands can only be tracked if they can be seen by enough cameras. If an object shall be translated a long distance the hands have to be moved an according way in the working volume. If in this case the user grabs, holds and moves an object with a large space between her/his hands, the range of available translational movement becomes even smaller and one hand often leaves the working volume unintentionally. Therefore, we prompt a hint suggesting the user to decrease the distance between the hands if one hand leaves the working volume and the distance between the hands was higher than $30cm$ in the preceding frame. This threshold value was used, because we observed that larger distances do not significantly improve the accuracy of object manipulations.

8.1.4 Experiments

We conducted an experiment to evaluate the performance of the proposed technique for two-handed grabbing and releasing virtual objects. Our hypothesis was that our method would be superior to other two-handed techniques commonly applied for hand-tracking devices. Moreover, we expected that our technique would generally be superior to one-handed techniques in high precision tasks.

In order to investigate these hypotheses, we compared our two-handed grabbing and releasing technique to three other approaches: the exploitation of a grabbing posture for triggering grabbing/releasing (i.e. only if the subject forms the grabbing posture with one or both hand(s) the virtual object moves according to her/his hands), the use of a standard 6 DOF controller (3D mouse) and the application of our recently developed one-handed technique; the Jerky Release technique (see Sec. 7.2). Note that in a short user study this one-handed technique outperformed a grabbing posture based technique and showed similar results as a standard 3D mouse in 6 DOF manipulation tasks.

Experimental Setting

Two connected PC-based systems were used in the experiment, one coupled to the cameras for tracking the hand (see Sec. 2.5) and another (Intel E6600, Geforce 8800) for running the virtual environment application. The application was visualized on a standard 19" TFT-Display. Additionally, a 3D connexion SpaceNavigator was connected to the second PC.

An illustration of how this system works together with our bi-manual technique is given in Fig. 8.6).

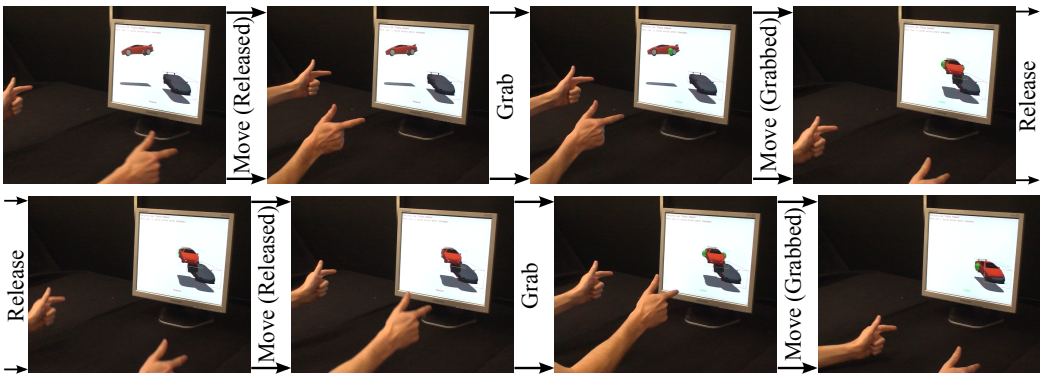


Figure 8.6: Image sequence of our system. Based on our bi-manual symmetric interaction technique a virtual object (red car) can be grabbed (move hands together), manipulated (as if gripped between hands) and released (move hands apart).

Experimental Task

In the experimental task a virtual object had to be moved to a specified position and orientation (see Fig. 8.7) by using the different 6 DOF techniques/controllers. The time needed for completing one task was split up into the approach time and

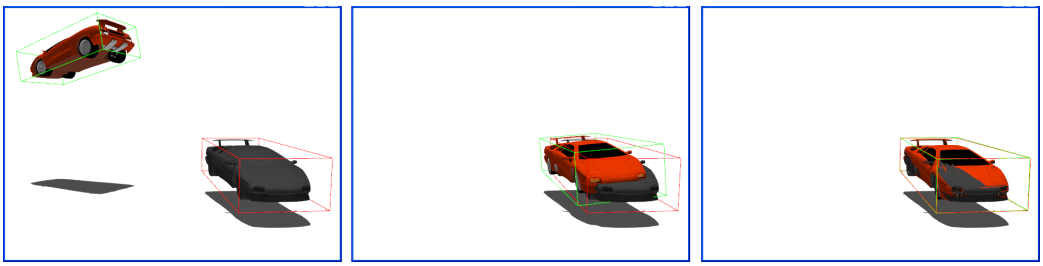


Figure 8.7: Illustration of the experimental task that had to be solved. Left: Initial situation. Middle: Roughly approached target pose (the approach time is measured). Right: Solved task (the precision time is measured).

the precision time. The approach time is measured when the object is approximately approached (see Fig. 8.7(Middle)); when it is moved roughly to the target

pose (i.e. less than 6 degrees rotational error and 4 units translational error). The period from this moment on until the user releases the object almost precisely (i.e. less than $1,5$ degrees rotational error and $0,5$ units translational error) at the target pose (see Fig. 8.7(Right)) is defined to be the precision time. This way, both the capabilities of performing rough/spacious as well as precise/fine manipulations are measured for each technique/controller. To reach the target pose several grab and release cycles had to be performed for the hand-tracking based techniques. No snapping algorithm (the object snaps to the desired pose, when it is near by) was applied.

Participants

Ten participants (all males, all university students) took part in the experiment. They had little or no virtual reality experience.

Procedure

Each participant had to solve the task four times by employing each of the four controllers. Thereby, the sequence of the employed controllers was permuted according to ten columns of three balanced mutually orthogonal Latin squares of size 4×4. All tasks had to be finished until the next controller was adopted. Before starting the test for each controller, its mode of action was explained and the subjects could familiarize with it in a short preparation time (two minutes).

Results and Discussion

The average approach and precision times for all individual subject are depicted in Table 8.1 and the total average times including standard deviations are illustrated in Fig. 8.8.

The high standard deviations in Fig. 8.8 are caused by two major reasons:

1. Different manual skills of the individuals. Most subjects that performed *slowly/fast* with a certain controller performed *slowly/fast* with the other controllers, too. Thereby, *slow/fast* means having high/low average per controller timings with respect to the other participants.

2. Different individual controller preferences. While several users could easily handle one controller some others were nearly incapable to work with it. In particular precisely positioning by using the 3D mouse had very inconsistent results. For this reason we argue that in an end user interface redundancies between different techniques (if possible) and/or selectable alternatives should be available.

Despite the high standard deviations, some statistical significances could be derived by performing the Student's t-test. In the following discussion we consider a confidence greater than 95% to be significant.

Roughly approaching an object was superior if only one hand or the 3D mouse was employed instead of a two-handed technique. This is mainly because more grab and release cycles were used for two-handed translational movements due to the higher need of workspace (both hands have to be located in the working volume) and the slightly reduced anatomical range (the shoulder has to remain straight if both arms are stretched). Only marginal difference was found between the 3D mouse and the single-handed object approaching. However, all techniques have significant superior results to the two-handed posture based technique. We observed the subjects to have some problems using different postures for grabbing and releasing, primarily because a higher concentration was needed for switching between the standard and the grabbing posture. The differences between the one-handed techniques and our two-handed technique is only significant with a confidence greater than 90% for the 3D mouse and 75% for the single-handed technique.

The best results in precisely positioning an object were achieved applying our two-handed technique. Furthermore, the differences between our two-handed technique and the two one-handed techniques are significant. The difference between our two-handed technique and the two-handed posture based technique are

Subject	**Two-Handed**		**3D Mouse**		**Single-Handed**		**Our Technique**	
	AT	PT	AT	PT	AT	PT	AT	PT
1	12.8	7.8	7.7	20.6	4.2	11.8	8.5	6.5
2	33.4	32.0	12.7	22.6	10.7	53.0	22.4	15.8
3	54.9	29.4	11.6	35.3	18.7	30.1	23.2	23.1
4	40.2	28.7	5.8	21.4	8.6	19.8	13.0	10.9
5	45.5	63.9	7.5	24.6	15.3	16.6	18.9	13.8
6	85.3	73.0	14.3	52.6	15.2	88.0	17.4	20.1
7	27.9	22.6	20.9	50.3	21.7	41.6	29.5	23.3
8	44.9	44.9	11.1	37.3	24.5	65.4	29.3	23.7
9	90.0	57.0	35.8	47.7	32.8	71.2	64.4	35.4
10	22.0	13.2	5.4	37.7	14.6	32.3	19.4	26.1
Mean	45.7	37.3	13.3	35.0	16.6	43.0	24.6	19.9
Total	82.9		48.3		59.6		44.5	

Table 8.1: Mean approach times (AT) and precision times (PT) for all individual subjects (in seconds). Note that our technique also performed best for the entire task.

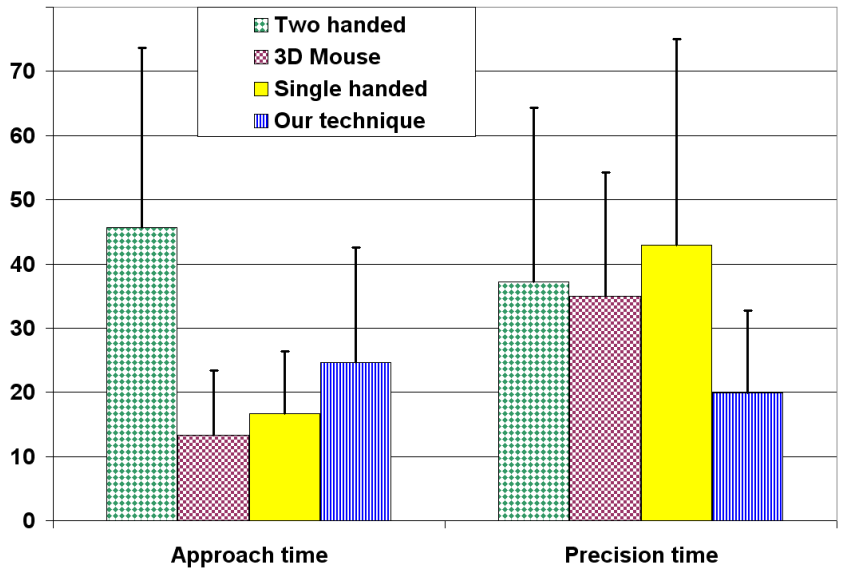


Figure 8.8: Completion times (in seconds) and standard deviations (error bars).

only significant with a confidence greater than 90%. Surprisingly, the precision times of the two-handed posture based technique are similar high as of the 3D mouse and the one-handed technique, respectively. This is caused by overhasty hand movements during a release operation while the hands were still holding the virtual object. Note that in this case the involuntary object movement leads to loosing the target pose so the subject must repeat the high precision positioning.

After the user experiments, we questioned the subjects for their subjective impression concerning the performance of the different controllers. For precisely positioning eight participants would prefer our two-handed technique, one would prefer the two-handed posture based technique and another one the 3D mouse. For approaching the target they rated both the single-handed technique and the 3D mouse best with four votes for each. The other two participants voted for our two-handed technique. Note that these ratings correspond to the measured times.

CHAPTER 9

RELATED AND CONCURRENT WORK

The large amount of literature in interaction techniques makes it practically impossible to give a full review of the previously reported methods here; elaborate analysis can be found in [BKLP05], or [JS07] for multi modal interaction. We will only discuss the most related methods that are designed for or can be applied to interaction based on markerless hand-tracking as an input device.

9.1 Menu and Object Selection

The task of selection is typically solved by two subtasks: positioning a cursor above an object or item and performing a clicking operation to finally select it. Thereby, both subtasks should be realized in an intuitive or at least easy-to-learn and easy-to-use manner. This has a great impact on the performance of an interface as selection operations are typically needed very frequently. However, designing suitable techniques for selection is a non trivial problem particularly in the context of bare-handed interaction.

9.1.1 Cursor Positioning

In selection tasks positioning a cursor describes the act of pointing to different graphical elements such as menu items, buttons or objects. Thereby, the choice of the positioning/pointing technique depends on the employed controller device (e.g. standard 2D mouse, touchpad, 3D mouse, 3D hand-tracker) and application (e.g. 2D or 3D, menu or object selection). For example for a standard 2D computer mouse a relative incremental position control (the cursor position is incremented relatively to the device movement when it touches the surface) with clutching (lifting the device de-clutches virtual from real movement) is doubtlessly the best solution. This way, the cursor can be moved incrementally to every position in the visual space by moving the cursor step by step. This also enables using a cursor speed dependent acceleration scheme such as the Pointer Ballistics for Windows XP for achieving faster as well as more precise cursor movements. This is the most common pointing facilitation technique for standard

2D mouses and touch pads and in fact can be considered as a final solution for such devices.

However, having tracked the global pose of both hands, pointing can be solved in various ways. The typical solutions for controlling a 2D cursor are either using the pointing direction of one hand or two dimensions of the hand position. Examples for using the hand position are the *Vision Based Touchpad* [ML04], where a planar surface on the table is used for simulating a touchpad, and employing the vertical and horizontal hand movement as done in [GFGB04] to design a non-contact mouse for surgeon-computer interaction. In hand-tracking interfaces that also provide the hand orientation, 2D cursor positioning is most commonly solved by the virtual pointer metaphor (see Sec. 6.2.2); the cursor moves according to the pointing direction of the hand. This turned out to be very intuitive and fast. In virtual environments, positioning of a 3D cursor is nearly always solved by using the virtual hand metaphor (see Sec. 6.2.1); a 3D cursor moves according to the 3D position of the hand. However, independent from the adopted positioning technique, the accuracy and performance both of 2D and 3D pointing is restricted due to natural hand tremor, tracking inaccuracies and the limited fine motor skills of humans in general. Therefore, various techniques for pointing facilitation were introduced to improve pointing performance and precision.

We classify these approaches into three categories indicating how the pointing facilitation is achieved: interface redesign, interface dependent facilitation and interface independent facilitation. In the next paragraphs the most related previous approaches will be reported according to these categories as shown in Fig. 9.1. As our technique belongs to the third category, we will discuss the first two categories only shortly. Readers being interested into a more comprehensive discussion of previous work we refer to [Bal04] and [BBM+06].

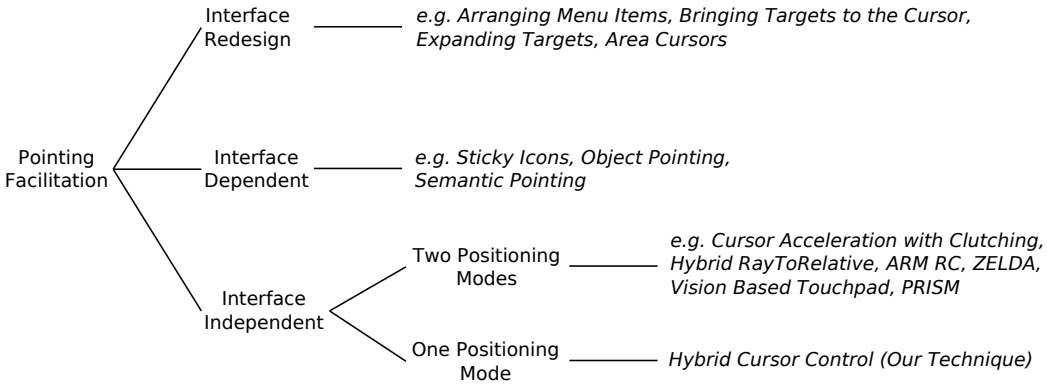


Figure 9.1: Categorization of pointing facilitation techniques.

One way to facilitate virtual pointing is altering the position or width of the

graphical elements (i.e. interface redesign). Examples are pie-menus arranged around the cursor [CHWS88], temporarily bringing virtual proxies of potential targets towards the cursor [BCR+03], dynamically expanding targets or the region in focus (e.g. [MB05, CB06, CK03]) or using area cursors (e.g. [KB95, WWBH97]). However, most of these techniques can only be applied to very specific settings and lead to drastic redesign of the interface. Moreover, these techniques can induce disorientation and are not generally applicable.

The second way of pointing facilitation is changing the mapping from the motor space (physical movements) to the visual space (cursor movements). This mapping is typically described by the display-control (D/C) ratio which determines how the controller movement $D_{controller}$ is mapped to cursor movement D_{cursor} with $D_{cursor} = \frac{D}{C} \cdot D_{controller}$. By adjusting the D/C ratio the sensed target distance and width can be changed without changing the overall visual appearance of the interface. E.g. decreasing D/C slows the cursor down which enlarges both width and distance in the motor space. In this case, the precision is increased at the cost of rapidness. The challenge is developing a dynamically changing D/C ratio which both decreases the target distance and increases the target width in the motor space. These techniques can be split up into interface dependent and independent techniques (see Fig. 9.1).

Interface dependent techniques adapt the D/C ratio dependent on the graphical element of the interface that is currently located below or near the cursor. Examples are *Sticky Icons* (cursor movement is decelerated above targets) [WWBH97], *Object Pointing* (cursor jumps over empty spaces) [GBBL04] or *Semantic Pointing* (cursor is accelerated above empty spaces) [BGBL04]. However, these techniques are driven by the arrangement of the interface. E.g. if large parts of the visualization are covered by selectable objects or items the cursor movements are slowed down permanently and selection can become awkward.

The techniques belonging to the category of *interface independent* pointing facilitation (see Fig 9.1) aim at facilitating cursor positioning independent on the arrangement of graphical elements. These approaches commonly adopt or assume two positioning modes which can be chosen by the user. A typical solution is using a clutching mechanism which enables one cursor movement mode and another de-clutched mode where the cursor is stopped. However, designing a suitable clutching mechanism for devices such as bare-hand-tracking is difficult. For example using different hand postures for clutching [VB05] turned out to be uncomfortable and difficult for many people due to high demands for the fine motor skills. Moreover, dependent on the adopted positioning technique (e.g. virtual pointer, see Sec. 6.2.2) the application of a clutching mechanism can generally degrade intuitiveness and user acceptance. Therefore, in recent years several different approaches were introduced, which instead incorporate one mode for coarse positioning and another mode for fine positioning.

For instance *Hybrid RayToRelative* [VB05] and *ARM RC* (Absolute and Relative Mapping Ray Casting) [KSMB08] use an absolute positioning mode (the cursor is placed where the hand points to) and a relative mode (the cursor movement is slowed down to gain precision). The two approaches vary in the way of switching between the two modes. In Hybrid RayToRelative two different postures of one hand are employed while in ARM RC pressing a button on a second device in the non-dominant hand switches the positioning mode. Malik et al. [ML04] introduced the *Vision Based Touchpad*, where the positions and postures of both hands are used to facilitate pointing on a large display. Therefore, a planar rectangular region on the table in front of the user is split up into two subregions, one for the left and the other for the right hand. With the non-dominant hand a window can be positioned on the display in an absolute manner (the subregion is linearly mapped to the entire display) by touching the planar surface. Simultaneously, the position of the dominant hand inside its subregion is linearly mapped to the window on the display, thus enabling fine positioning a cursor inside this window. Clicking is achieved by touching the surface with the right index finger. Based on this approach *ZELDA* (Zoom for Enhanced Large Display Acuity) [KSMB08] was introduced. But in contrast this technique uses a distant pointing device in each hand. Therefore, buttons are used instead of touching the planar surface. Furthermore, a zoom-in of the window can be visualized and its size and zoom factor can be adjusted using a scroll wheel on the device in the non-dominant hand. A different pointing facilitation technique is *PRISM* (Precise and Rapid Interaction through Scaled Manipulation) [FKK07]. PRISM consists of two major positioning modes: positioning with speed dependent cursor acceleration and an offset recovery mode where cursor movements are stopped. Thereby, the activation of this offset recovery mode depends on the direction of movement; only if the cursor is moved toward its position in the motor space. Unfortunately, all these approaches suffer from two mutual problems: first, as these techniques are quite complex (different positioning modes and/or use of different buttons or postures) every user needs an introduction and some training for learning to handle it. This reduces the user acceptance. Second, the switching between two different positioning modes reduces the performance due to the need of extra time for switching and/or reorientation.

Note that the reported techniques does not generally exclude each other. Especially, techniques belonging to different main categories (see Fig 9.1) are suited for being used together. Further note that the techniques belonging to the interface independent approaches are by definition the most general solutions to pointing facilitation.

9.1.2 Bare-Handed Clicking

As soon as the cursor is properly positioned above a target a suitable clicking technique is needed to complete the selection operation. If a hand-tracking device is employed, the clicking operation has to be simulated, because no physical buttons are available. Note that for selection purposes exploiting a DOF of the hand pose for triggering clicking events is feasible because in general not all 6 DOFs are needed for positioning the cursor. In the following the different approaches for clicking simulation suitable for bare-handed interaction are outlined.

The first and easiest solution for performing clicking operations is extending the hand-tracking interaction interface with additional physical buttons as for example floor pedals. However, this kind of interaction turned out to be awkward and slow (according to [GFGB04]).

Another approach is using a cursor dwell time threshold for triggering a click event as for example used in [WP03] and [GFGB04]. Although this is simple, it introduces a constant lag in the interaction.

A further approach is to use speech to signal a selection [Bol80]. But this is especially excessive if several click down and up events have to be captured.

To perform a click by specified movements of the hand is another option. In [GFGB04] clicking is performed if the user moves her/his hand 20 cm toward the camera. We observed this technique to lack efficiency and comfort, because it requires a quite spacious hand movement. A better solution is proposed in [VB05], where clicking can be performed by a small movement with the index finger, similar to how we move when clicking a physical mouse button. But obviously, this technique can not be used when the pointing direction of the index finger is needed at the same time (e.g. for cursor positioning employing the virtual pointer metaphor, see Sec. 6.2.2).

An additional commonly used technique is exploiting different hand postures to click. In [GWB05] and [VB05] a button down or up event is triggered, when the thumb is moved in or out toward the index finger side of the hand. Unfortunately, this often leads to unmeant changes of the pointing direction due to unstable tracking states during the transition between two postures. Moreover, even changing between simple postures is significantly more complex and uncomfortable than simply pressing a mouse button.

9.2 3D Object Grabbing and Releasing

In contrast to cursor positioning in selection tasks, the extend of the virtual working space is typically not finite in 3D object manipulation. Nevertheless, precise manipulations of a selected 3D object should be possible at any location. To

this end, a mechanism for grabbing and releasing the object is needed in order to manipulate it step by step. Additionally, this mechanism must enable precise releasing it in the desired pose.

Different manipulations can be applied. For example during complex object movements all 6 DOFs of the user's hand pose are used to determine the pose of an object. Therefore, no DOFs are available for triggering grabbing/releasing and the virtual clicking techniques can in general not be applied to solve this problem. Other solutions have to be found for determining when the object shall be attached to the hand (i.e. coupling the object's movements to the hand's movements).

According to Zachmann [Zac00] grabbing an object (i.e. attaching the object to the hand) can be realized in (at least) three different ways: single-step, two-step or naturally. Single-step grabbing attaches the object at a certain event (e.g. a spoken command like "grab thing"). Two-step grabbing can be further divided into the following interaction steps:

1. Some event (e.g. a posture or spoken command) switches the grabbing mode on; only in this mode, objects can be grabbed.

2. The object is attached to the hand at another event.

To release the object usually the same event as in the first step is used. In the grabbing mode natural grabbing is typically realized by conditioning collisions of virtual hand representations with the object (e.g. two virtual hand models must collide with the object). The object's movements will be coupled directly to the hands', when the object is touched this way. Note that if the virtual scene is rendered into the hands' working area/volume the virtual hand representations are normally not visualized, because the real hands can represent themselves. Otherwise, the virtual representations of the user's hands have to be rendered. However, in both cases precise two-handed manipulation of a small object is nearly impossible, because the distance between the hands has to be very small for grabbing and manipulating the object.

Using physical buttons, a dwell time threshold or speech for triggering an attach action suffer from the same drawbacks as in the case of clicking. As well exploiting one DOF of a hand pose is only possible for two-handed manipulation, because for object movements normally all 6 DOFs of one hand are needed.

Therefore, most approaches adopt grabbing postures to determine whether an object is attached to the hand or not (e.g. [MF04] or [BI05]). Unfortunately, it turned out to be quite difficult to release an object at a precise position [Osa06]. The reasons for this are: first, it is demanding for people to fix their hands precisely in midair without having physical support. Second, judging the release point without tactile feedback can be difficult. Third, the finger movements of a grabbing action often induce involuntary changes of the hand's global pose. To

solve the third problem of using grabbing postures Osawa [Osa06] proposed an approach to automatically adjust the release pose of a virtual object based on the relative speed of the two grabbing fingers (usually the thumb and one forefinger).

Furthermore, in current markerless hand-tracking systems most of the different postures humans exploit for grabbing (e.g. the 3-point pinch grab exploiting the thumb and two other fingers or the power grab exploiting the whole hand [Zac00]) are not available, typically only one grabbing posture is supported per hand. Additionally, in order to ensure a stable tracking, this posture must be formed very exactly and clearly. We observed this to be cumbersome for most users.

9.3 Bi-manual Techniques

We focused on bi-manual symmetric manipulation, therefore, we will not discuss asymmetric techniques here, an overview of asymmetric interaction can be found in [BKLP05]. In the following the symmetric 3D interaction techniques most relevant to our work are outlined.

Two-handed scaling of objects is a popular example for symmetric bi-manual interaction. This can be solved as follows: the user picks up two sides of an object and can scale the object by moving her/his hands apart or together (e.g. [ZFS97], [ML04]).

As well traveling in the virtual environment can be solved bi-manually symmetric. In the Polyshop system [MM95] a user can travel by performing a rope gesture. By pulling on an invisible rope with both hands the user can pull herself/himself through the environment.

Using both hands for moving an object is most related to our work. Cutler et al. [CFH97] used datagloves to track the hands' movements for direct manipulation on a responsive workbench. In this context they introduced several bi-manual symmetric techniques: the grab-and-twirl technique enables the user to pick up two sides of an object and then to carry and turn it around with both hands. By fixing one or more DOFs of the applied transformation several other techniques were derived such as the grab-and-carry technique (no roll around the line connecting the two hands is allowed) or the turntable technique (only turning around a fixed axis of rotation is allowed). These techniques turned out to be both intuitive and efficient. However, these techniques implicitly assume the employment of the virtual hand metaphor for determining the two pivot points (i.e. the two points where the object is grabbed). Therefore, it is not suitable to manipulate small objects as in this case the distance between the hands has to be too small for precision.

Part III

Interaction Interfaces and Applications

CHAPTER 10

OWN INTERFACES AND APPLICATIONS

With a novel interaction device typically novel kinds of interaction interfaces and applications become possible. Their careful development and investigation is a crucial step in HCI. We implemented several interfaces and applications for bare-hand-tracking. In this context both existing as well as novel interaction techniques (see Part II) are employed.

10.1 Virtual Building Blocks

We developed an interface for primitive assembly tasks, where the position and orientation of a virtual robot hand is directly deduced from the user's hand. Rectangular solids can be grabbed by moving the robot hand close to it and switching from the 'pointingA' posture to the 'picking' posture. While retaining the 'picking' posture, the robot hand as well as the grabbed solid can be moved to another position and orientation. Switching back to posture 'pointingA' indicates the robot hand to release the solid. For an illustration see the accompanying video and the exemplary image sequence depicted in Fig. 10.1.

By switching to the 'pointingB' posture, the viewing point and direction of the virtual camera can be modified. Thereby, the pointing direction of posture 'pointingB' determines the viewing direction that should be chosen to look at the scene. For example, if the index finger points towards the left, the virtual camera looks on the scene from the right. The position of the user's hand determines the translational shift with respect to the current viewing direction (moving the hand towards the right moves the virtual camera right too). Switching back to posture 'pointingA' or 'picking' quits the viewing modification mode.

This application was tested by many subjects, and all were able to grab the solids and stack them together accurately. Because the used postures are the same postures as in real life, the grabbing is intuitive. However, some subjects had problems to form the grabbing posture correctly. They first had to learn the specific finger coordination. Moreover, the needed concentration to perform grabbing and releasing hindered them to perform the remaining hand coordination for manipulation. This complexity is due to the high demands for the fine motor skills in

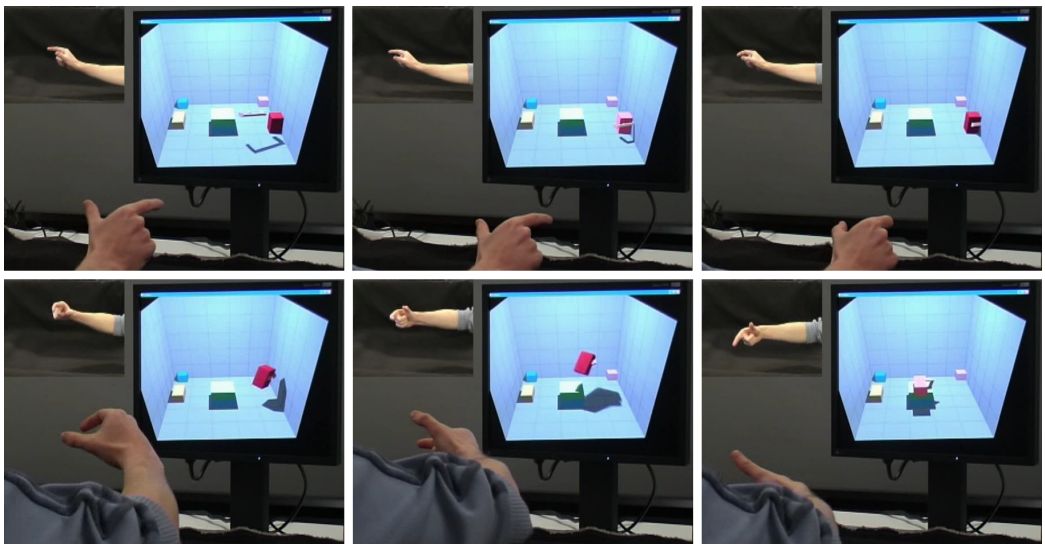


Figure 10.1: An exemplary image sequence (from top left to bottom right) of our virtual building blocks application. The red solid is grabbed and put onto the solids in the middle.

forming different postures. The feedback from this application was one reason for our investigations on alternative interaction techniques for solving the grabbing and releasing problem (see Sec. 7.2 and 8.1).

10.2 Mesh Editing/Deformation

Recent mesh editing applications to deform surfaces typically use several sets of mesh vertices as handles that can be globally translated and rotated. Meanwhile the surface in between is deformed by performing an energy minimization. In this context, the first task to solve is suitably selecting these sets of vertices. As the surface can be a complex 3D model, performing this selection task can be difficult with standard interaction devices such as a 2D mouse. Using the tracked poses of both hands, it is possible to use an intuitive asymmetric interaction metaphor, where the non-dominant hand is employed to move the 3D object while the dominant hand moves a virtual pen or similar to mark vertices on the object. This corresponds to painting on an object in real life (e.g. painting easer eggs).

Moreover, suitably articulating the 3D model by translating and rotating the handles is even more complex. To this end Llamas et al. [LKG+03] proposed to use a 6 DOF magnetic tracker in each hand. Thus, the full poses of two handles can be controlled simultaneously which, combined with a real-time mesh defor-

mation method, can be used for computer puppetry. As we are able to use the hands directly as 6 DOF input devices, we applied our bare-hand-tracking device to control such a modern mesh deformation method [PDK07] to establish a bare-handed computer puppetry application.

10.2.1 Vertex Selection

To select the mesh vertices comprising a handle, the tracked pose of one hand is exploited to perform a rigid body motion of the mesh while the other hand pose is used to move a ball of a certain radius. Meanwhile all mesh vertices within the ball are marked. By switching from the posture 'pointingA' to 'picking' and back again to 'pointingA' the mesh can be released or picked, depending on whether it was picked or released previously. Additionally, the hand controlling the selection ball can be used to handle a simple 3D menu. By moving the ball above a 3D button and performing the Roll Click (see Sec. 7.3) or a change to posture 'pointingB' the button is pressed. This way several fundamental settings/commands can be modified/issued (e.g. increase/decrease ball radius, switch to deformation mode, mark vertices for the other handle). For an illustration see the accompanying video and the exemplary image sequence depicted in Fig. 10.2.

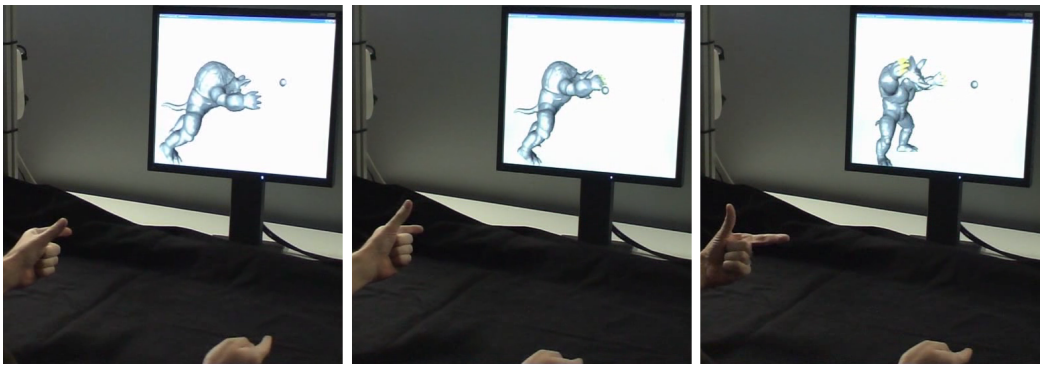


Figure 10.2: An exemplary image sequence of the selection mode in the mesh editing application. The vertices belonging to one handle are selected on the fingers of the armadillo model.

10.2.2 Computer Puppetry

To be able to intuitively change the position and orientation of two handles at the same time, we use the tracked pose of each hand to directly deduce the pose of one handle. If for example the left hand is moved upwards, the corresponding

handle is translated upwards accordingly. Equally the orientation of the handle can be changed by rotating the hand around its hand center. Similar to the case of marking vertices, shortly pausing in the 'picking' posture results in releasing or picking a handle. This combination of real-time mesh deformation and hand-tracking can be exploited as a nice computer puppetry application. For example intuitive real-time gesticulations of the armadillo object are possible by coupling each hand of the armadillo object to a hand of the user. For an illustration see the accompanying video and the exemplary image sequence depicted in Fig. 10.2.

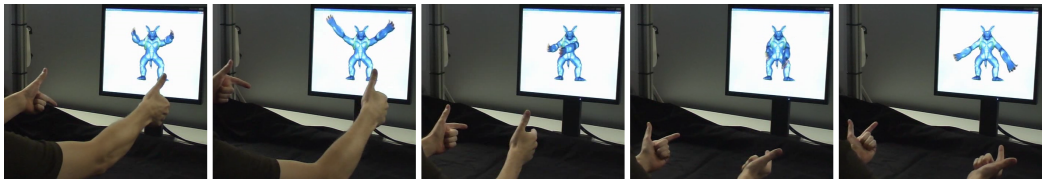


Figure 10.3: An exemplary image sequence of the puppetry mode in the mesh editing application. Each user hand controls one hand of the armadillo model. The mesh in between the hands and the shoulder of the armadillo model is deformed in real-time.

Due to its naturalness this way of interaction is very easy to handle, although the amount of controllable degrees of freedom is high (two 3D translations and rotations). It is tedious to perform such manipulations using traditional types of controllers (e.g. 2D/3D-mouse or joystick).

10.3 Virtual Pointer

To enable menu navigation and control we coupled the standard 2D mouse cursor to our tracking device. Therefore we employ the Hybrid Cursor Control technique introduced in Sec. 7.4 combined with the virtual pointer metaphor (see Sec. 6.2.2). By pointing in different directions forming the 'pointingA' posture, the mouse cursor moves accordingly. Using the Hybrid Cursor Control precise as well as fast point and select actions are possible.

For simulating the left and right mouse buttons we used the Roll Click technique introduced in Sec. 7.3. We also experimented with other solutions as for example exploiting posture changes for triggering button events. By changing from posture 'pointingA' to posture 'pointingB' a button down event is indicated and changing back again to posture 'pointingA' led to a button up event. Unfortunately, the posture based approach turned out not to be applicable mainly due to comfort issues, because changing postures is more demanding for the fine

motor skills. In particular, several successive button down and up cycles led to unpleasant fatigue of the thumb and the front finger in our experiments. In contrary adopting the Roll Click did not have these effects.

This virtual pointer interface was tested by many subjects. All could handle it easily and were able to perform precise as well as fast operations. For an illustration see the accompanying video.

10.4 3D Manipulation Interface

When the user's hand is directly used as an input device for controlling a 3D object manipulation application it is extremely desirable that no other controller is involved during appliance to ensure liquid interaction. Therefore we designed a graphical user interface (GUI) that is fully controlled by the user's tracked hands. In our GUI a 3D scene is shown and several basic manipulation tools can be selected from a toolbar, when the menu mode is active (see Sec. 7.6). The user can choose a tool by moving a hand model such that it intersects the 3D object representing a tool (currently a labeled cylinder), and performing a Roll Click (see Sec. 7.3).

Our system is designed as a state machine, where the states are represented by the different manipulation tools. If a certain manipulation state is occupied and the manipulation mode is active, specific manipulations are applied to the selected objects (e.g. state 'Move' for translating and rotating objects) or the virtual camera (in state 'Steer'). In the following the different available manipulation states are described.

In the 'Move' state the currently selected objects are translated and rotated according to the user's hand movements as long as they are grabbed, which is determined by the Jerky Release technique introduced in Sec. 7.2. In case that currently both user hands are used for manipulation instead the Bi-manual Symmetric Grab technique (see Sec. 8.1) is used for grabbing, moving and releasing the selected objects. This way, the precision and control over the object manipulation can be increased.

In the 'Scale' state the object(s) is(are) scaled up if the user moves her/his hand to the positive x-direction and down if she/he moves her/his hand to the negative x-direction. In case that currently both user hands are used for manipulation, instead of the movement in x-direction the distance between the hands is taken into account. Increasing the distance scales up and decreasing the distance scales down the object(s). In both cases no scaling is applied while the hand movement is classified to be fast and jerky according to the technique in Sec. 7.2. This way, the object can be scaled up or down arbitrarily in several cycles.

In the 'Select' state the two most common standard techniques (according to

[PIWB98]), namely the virtual hand metaphor (an object can be selected when it collides with a virtual hand) and the virtual pointer metaphor (an object can be selected when it collides with a virtual ray emanating from the virtual hand) are available. Each technique is combined with our clicking technique (see Sec. 7.3). When the virtual pointer metaphor is applied an additional ray is drawn in the visual feedback illustrated as a simple line emanating from the hand model toward the pointing direction. For an illustration see the two accompanying videos and the exemplary image sequence of selection and manipulation depicted in Fig. 10.4.

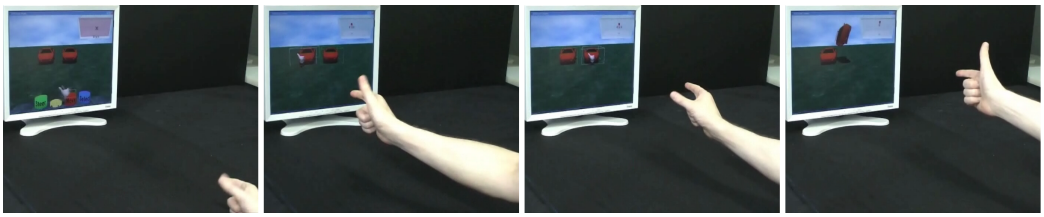


Figure 10.4: An exemplary image sequence of our 3D manipulation interface. An object is first selected and then moved.

In the 'Steer' state the virtual camera can be moved. Therefore we use the traveling technique described in Sec. 7.5. This control mode enables the user to quickly reach every perspective in the scene by controlling 6 continuous DOFs. While the pose of one hand already provides 6 DOFs, two hands can further support the steering of the virtual camera. Therefore, we additionally implemented bi-manual symmetric steering of the virtual camera similar to the Two-Handed Flying in [MFPBS97] as an optional mode. It is activated as soon as the second hand enters the working volume.

10.5 Joystick Device

In order to connect a game to our hand-tracking device the pose and posture of the hands have to be mapped to specific controls in the game. To this end, a good solution is using a virtual joystick driver due to the joystick support in most games. The virtual joystick driver receives information from the hand-tracking device about the hands' poses and postures. It simulates a physical controller allowing for mapping real hand movements to specific actions in the game. To implement such a driver we used the freeware program Parallel Port Joystick (Version 0.83) of Deon van der Westhuysen. This framework provides a virtual joystick driver for Windows 32 with up to 8 analog controls (finite continuous values) and 32

digital buttons (boolean values) and allows for coupling them to arbitrary hand movements. In the following two subsections, we describe which analog and digital controls are available in our current implementation. However, which hand movements are mapped to which controls depends on the current game and interaction mode.

10.5.1 Analog Controls

The analog controls can be coupled to the coordinates of the hand positions, to the pitch, yaw and roll angles of the hand orientations and to possible combinations of these variables. The spatial context of hand and display is illustrated in Fig. 10.5. Moreover, combinations of both hands' poses can be used. For two-handed analog

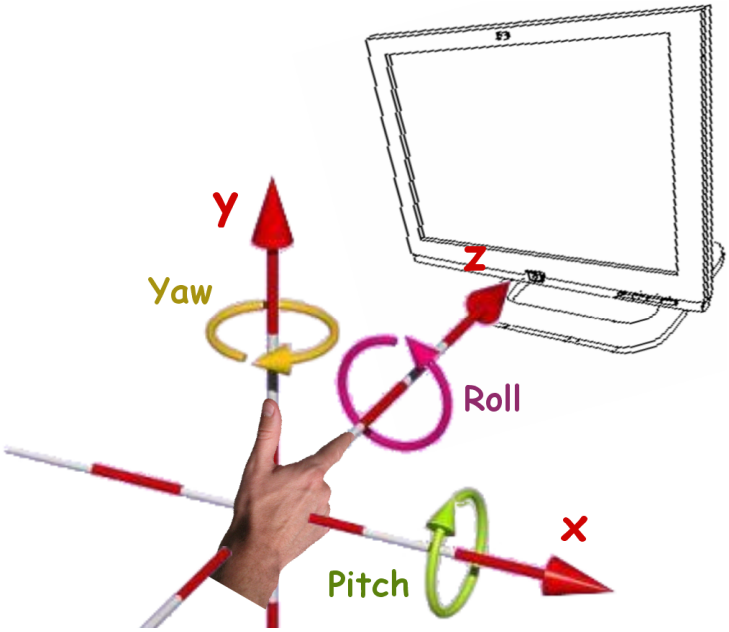


Figure 10.5: The spatial context of hand position, hand orientation and display. The x, y and z-coordinates as well as the pitch, yaw and roll angles can be mapped to analog values of the joystick device. In the shown hand orientation all angles are zero.

input we implemented the steering wheel gesture [CFH97], where the position is determined by the average hand positions and the orientation is determined by two different perpendicular orientations: the orientation of the line connecting the two hand positions and the average hand orientations with respect to this line.

The limits and deadzones of the respective joystick analog controls can individually be set. The lower and upper limits describe an interval in the hand

coordinate system and determine the mapping of the hand coordinates or angles to the deflections of the respective virtual joystick's analog control. Values less or equal than the lower limit are mapped to the maximal negative deflection and values greater or equal than the upper limit to the maximal positive deflection. For simplicity the maximal negative deflection is defined to be -1 and the maximal positive deflection to be 1. The center between the lower and upper limits is mapped to a deflection of zero. Dependent on the size of the deadzone also values close to the center are set to zero deflection. Additionally, it is possible to apply an arbitrary non-linear mapping to an analog control, which can simplify the handling in a game (e.g. steering in a racing game can be improved this way).

Mapping the 3D position to three independent analog values can be solved straightforwardly by using the X, Y and Z coordinates of the hand positions. To split the 3D orientation into three independent angular values, we first exploit the horizontal angles to determine pitch and yaw. Thereby, pitch is defined to be the elevation angle and yaw the azimuth angle. Both angles are zero if the hand's index finger is pointing toward the screen. Note that this way the pitch angle α_p is restricted to the interval $(-\frac{\pi}{2}; \frac{\pi}{2})$ and the yaw angle α_y to $(-\pi; \pi)$. Second, the roll angle α_r is determined by the rotation around the pointing direction. We define it to be zero if the thumb's pointing direction is parallel to the plane spanned by the up-vector $(0, 1, 0)$ and the index finger's pointing direction. Note that α_r is also restricted to the interval $(-\pi; \pi)$.

This way, the angles of the hand orientations are unambiguously defined. In case that the index finger nearly points up, the computation of the yaw and roll angles gets unstable, however, in practice this rarely happens.

The world coordinate frame is oriented such that the positive X-axis points from left to right, the positive Y-axis from bottom to top and the Z-axis from the user to the screen (see Fig. 10.5). The pointing directions of index finger and thumb can be approximated by applying the hand(s) rotation to the vectors $(0, 0, 1)$ and $(0, 1, 0)$, respectively, because the identity rotation describes the hand pointing toward the front with the thumb pointing upward. Note that in the two-handed case (using the steering wheel gesture) the orientation is described by the identity when the Y and Z-coordinates of both hand positions are equal and the average orientation around the connecting line is the identity. In this case, the angles can be computed the same way.

10.5.2 Digital Controls

A virtual digital button can either be pressed or not. We can map arbitrary functionalities to button events. Besides mapping different postures to different buttons we exploit various hand gestures determined for example by thresholding on the hand position or velocity with respect to one coordinate or angle. Moreover,

the button states determined by the Roll Click technique (see Sec. 7.3) and the air tap gesture (a button can be pressed by a small movement of the index finger, see [VB05]) are mapped to digital controls.

We distinguish two kinds of buttons in games: first, buttons that are used for infinitesimal events like "jump" or "reposition" and second, buttons that also need to be pressed a longer while like "crouch" or "brake". The first class of buttons can be assigned to all kinds of hand movements while the second class needs the ability to regulate the duration the button is pressed.

CHAPTER 11

COMMERCIAL APPLICATIONS

The investigation of how existing applications can be interfaced to a novel interaction device is also of crucial importance. Once an application is identified as a candidate for being controlled with a new device, the detailed design of the interaction has a great influence to the applicability and user acceptance. To this end, we investigated the interfacing of our device to two commercial applications and four 3D games.

11.1 Zugspitze 3D (Fly Through)

Zugspitze 3D is an interactive photorealistic 3D map released by RSS Remote Sensing Solutions GmbH in 2006. With a geometrical resolution about 900 times higher than in Google Earth the Bavarian alpine region can be virtually explored. The terrain data is annotated and many hiking tours can be selected and superimposed.

In this application two major modes of interaction can and should be addressed: the flight through the terrain data and the menu navigation. In this context, we experimented with different one-handed and two-handed solutions, which are described in the following subsections.

11.1.1 One-Handed

To control the fly through with only one-hand, we use the flying vehicle control metaphor [WO90] with the modifications explained in Sec. 7.5. The visual feedback box is integrated optionally; it can be activated by the user. Obviously, the additional visual feedback of superimposed 2D arrows can not be applied to a fly through application as it is not known which object or region the user is currently inspecting. For an illustration see the accompanying video and the exemplary image sequence depicted in Fig. 11.1.

Handing the menu of Zugspitze 3D is performed by controlling the 2D mouse cursor with the virtual pointer technique described in Sec. 10.3. Switching between the fly through and menu navigation mode can be achieved by changing to

Figure 11.1: An exemplary image sequence of controlling the Zugspitze 3D application.

posture ‘palm’ and back to ‘pointingA’ or ‘pointingB’ (see Table 3.1 in Chapter 3).

This interface to Zugspitze 3D was tested by a great many of people (several hundreds). All were capable to handle the flight through the terrain. Considering the fact that this interaction mode needs to handle six DOFs simultaneously, this is a very promising result. For an informal comparison we also connected a 6 DOF 3D mouse (3D SpaceNavigator) to Zugspitze 3D and asked several people to test it. However, the results were considerably inferior. Several people had severe problems handling the 3D mouse while some were completely incapable to use it without extensive training. Moreover, all testers liked the bare-handed approach more than using the 3D mouse.

11.1.2 Two-Handed

Having tracked both hands, we can handle both the fly through as well as the menu navigation at the same time. Even though it is difficult to perform complex flying tasks while simultaneously handling menu navigation, it can be of great advantage that the hand used for the fly through can stay in the fly through mode, because otherwise (if one hand is used for both) the user has to reorient herself/himself after every switching from the menu navigation mode. Therefore, the fly through functionality is mapped to one hand (normally the dominant hand) and the menu navigation functionality is mapped to the other hand (normally the non-dominant hand).

Although this interface could be handled by all subjects, most had difficulties handling the two modes at the same time due to the need of asymmetric coordination. Therefore, the profit of using this two-handed control is restricted and in consideration of the additional complexity for the user we think that this solution would not widely be accepted in practice.

11.2 RTT-DeltaGen (3D Manipulation)

RTT DeltaGen is a professional tool enabling real-time visualization and editing of professional CAD/CAS datasets. We integrated the hand-tracking device with Version 8.5 released 2009 by the Realtime Technology AG. One marked strength of using bare-handed tracking for 3D interaction is the ease and quickness that people become able to handle even complex tasks. Therefore, a scenario easily enabling normal people to interact with CAD models was the focus in the integration with RTT DeltaGen. Obviously, suitably editing CAD models needs a lot of practice, independent on the adopted interaction device. Therefore, we concentrated on the visualization part of RTT DeltaGen. The user should be able to easily inspect a 3D object (e.g. a car) from all viewpoints as well as to switch between different variants of the object (e.g. lacquer, rims) or activate predefined animations (e.g. open a door). The interaction should be as easy to handle as possible.

To this end, at least two different interaction modes are needed: one mode for 3D object manipulation/inspection, where the object or the camera moves according to the hand(s), and another mode like menu navigation, where the user can select a variant/animation and can issue a switch/activate command. In this context, enabling easy switching between these modes is crucial. Moreover, the user should always be aware of the mode she/he is currently in. To this end, the two modes should be distinguishable and easy to switch. We experimented with typical different purely one-handed solutions as for example different postures for different modes or splitting up the working volume in different regions. Using postures led to two main problems: the users were distracted by ensuring that they form the correct posture and the object movement could not be stopped precisely due to movements induced by the posture change. Splitting up the working volume often led to involuntary mode switching because no tactile feedback is available. Moreover, the working space for 3D manipulation is reduced in this case. Therefore, we also experimented with two-handed interaction and came to a solution, where one-handed interaction is used for menu navigation and two-handed for 3D manipulation. This solution performed best in several informal experiments and is implemented as follows.

If two hands are concurrently located inside the working volume, the Bimanual Symmetric Grab technique (see Sec. 8.1) is adopted to translate and rotate the 3D object. This way, the object can intuitively be inspected from any point of view. See Fig. 11.2 for an illustration.

In the one-handed interaction mode the user controls the 2D mouse cursor as explained in Sec. 10.3. She can click on object parts or other items to start animations or switch the design variant. This way also the menu of RTT DeltaGen can be handled in order to change arbitrary settings. In Fig. 11.3 two screenshots

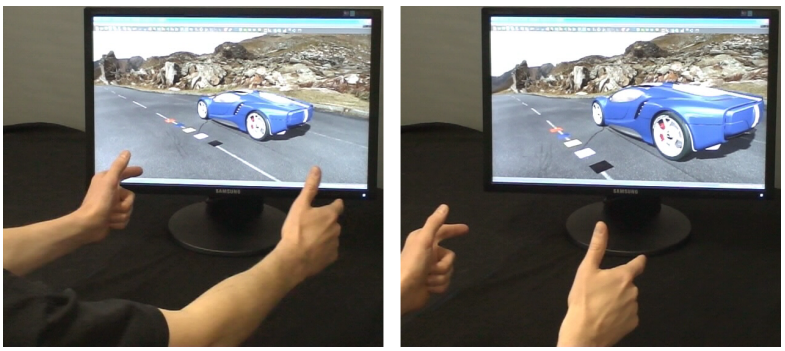


Figure 11.2: Two screenshots of using the Bi-manual Symmetric Grab in RTT DeltaGen. By moving the hands from the front (left image) to the back (right image) the car and surrounding is moved accordingly. (Car model courtesy of RTT AG)

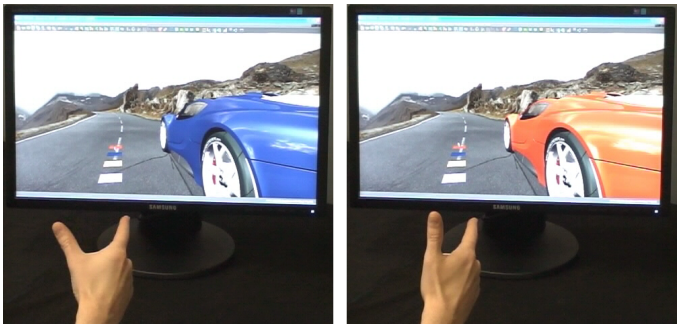


Figure 11.3: Two screenshots of using the 2D mouse cursor in RTT DeltaGen. By clicking (using the Roll Click technique) on the red squares plane (used as buttons) the color of the car is switched from blue (left image) to red (right image). (Car model courtesy of RTT AG)

illustrate such a clicking operation used for changing the lacquer of a car model.

Switching between one and two-handed interaction is solved according to the Bi-manual Symmetric Grab technique described in Sec. 8.1.2.

By using one and two-handed interaction for distinguishing cursor control and 3D manipulation, mode switching is easy and the user is always aware of the mode she/he is currently in. In addition, because only easily manageable interaction techniques are adopted, also unpracticed users are able to use it without training. For an illustration of the interface see the accompanying video.

11.3 3D Games

Our markerless hand-tracking based virtual joystick (see Sec. 10.5) was tested as an input device for three different representative 3D game genres: two racing games, a flight simulator and a first-person shooter. For each game we investigated different modes of interaction and present one or more suitable solutions. To analyze the performance and usability of different interaction modes we conducted an informal user study. Five university students (all male) took part in several experiments concerning the different games and different joystick configurations. The analyses of the different games and interaction modes are based on the comments of the subjects and our observations on how they were capable to handle it.

To avoid other controllers than our device being involved during interaction also menu navigation has to be addressed. This is solved by using the hand-tracking device to simulate the standard 2D mouse as explained in Sec. 10.3. In games not supporting mouse movements, small rotational hand movements around the pitch and yaw axis are exploited to move the current selection up/down and left/right. Clicking is also done via the Roll Click.

After investigating different possibilities of bare-handed 3D gaming, we designed and performed a more formal user study exploiting questionnaires and video analysis. All three game genres were tested and rated by 15 subjects using the bare-handed interaction mode that was previously found to work best. For the purpose of comparison, a standard interaction controller was also involved in the user study.

11.3.1 Racing Simulator

The first games we investigated are racing games named Re-Volt released by Acclaim Entertainment in 1999 and Torcs (The Open Racing Car Simulator) in Version 1.3.1 released in 2008, which is an open-source software available under the GNU general public license. In Re-Volt the user steers a toy car through everyday environments (e.g. a street or a garden) and can collect items (randomly assigned) as for example rockets or turbo boosts. The items can be activated by pressing a button. In this game the following digital controls are available: fire, flip car, reposition, pause and horn. In Torcs the player can choose between typical racing cars and can select different typical racing tracks. The relevant digital controls available in the game are: switching to ABS mode and turning the speed limiter on/off. Both racing games have two continuous DOFs: steering left/right and accelerate/reverse.

We experimented with different analog and digital mappings of hand movements to controls in both games. For one-handed left and right steering it turned out that only the X-axis coordinate of the hand position or the rotational yaw and

roll angles can be adopted in an intuitive and easy-to-use interface, because otherwise the direction of virtual movement does not correspond to the direction of real movement. For two-handed interaction the roll was by far the best choice for steering left/right. To further improve the steering movements (both for one and two-handed), a non-linear mapping is applied to the according analog control. To this end, the absolute values of the negative and positive deflections are raised to the power of 0.7. Note that the deflections have values between -1 and 1. This way, turning the car is more sensitive to small hand movements (simplifies fast reactions), but the impact of spacious hand movements is decreased (minimizes unintentional skidding of the car). Experiments with the subjects showed this to be controllable more easily.

For acceleration and reverse we identified two different possibilities which could easily be handled by the subjects: either using the Y-axis coordinate (the hand(s) has to be moved near to the user to reverse and vice versa to accelerate) or the pitch angle(s) with a range of 0 to $\frac{\pi}{2}$ (see Sec. 10.5.1). While using the Y-axis coordinate was a bit more intuitive than the pitch angle it unfortunately induced more fatigue due to the need of leaving the arm at a specified position to accelerate. In contrast, using the pitch angle with this limits was very comfortable because in a racing game the user normally accelerates most of the time, so the user's hand can mostly remain in the pitch center orientation.

For the digital controls we experimented with various configurations. However, for lack of space we only discuss our final solutions here. In our final one-handed solutions, which served best for most subjects in both games, we use pitch for accelerate/reverse and yaw for left/right. In the two-handed case we use the combined roll angle for left/right steer. The digital controls in Re-Volt are assigned as follows: a left roll click for fire, a right roll click for flip car, the 'palm'-posture (all fingers are stretched) for reposition, the 'pointingB'-posture (only index finger is stretched) for pause and a small and fast movement down for horn. In Torcs the ABS mode can be switched on/off via the Roll Click gesture and by shortly forming the 'pointingB' posture the speed limiter can be switched on/off. For an illustration of the one-handed solution for Re-Volt see the accompanying video and the exemplary image sequence depicted in Fig. 11.4.

Both solutions were found to be intuitive for both games. All subjects were able to use it without a long familiarization. They liked to use this kind of interaction for racing games. However, using both hands turned out to be more exhausting due to the need of holding both hands in midair. On the contrary, the one-handed technique induced only little fatigue because the hand and arm could rest on the table.

Figure 11.4: An exemplary image sequence of playing Re-Volt using our interface. Performed actions from left to right: turn right, turn left, drive straight, flip car, accelerate.

11.3.2 Flight Simulator

In order to investigate the capability of bare-handed input for a flight simulator, we connected our joystick device to the game Yager of Yager Entertainment released in 2003. In this 3D game a futuristic fighter jet has to be controlled, so military actions are also part of the game. Yager offers two modes of operation called "jet-mode" and "hover-mode". We focus on the first one, since it provides a control similar to most other flight simulators where typically throttle, pitch, yaw and roll have to be assigned to the input device(s). Obviously, the most intuitive mapping is assigning these controls to the corresponding hand movements. Therefore, the hand pitch, yaw and roll are used for the plane's pitch, yaw and roll controls and the Z-coordinate of the hand(s) position is used for throttle. This approach is similar to the typically adopted control in fly through applications (see Sec. 11.1). Resulting from several experiments with the subjects this solution was found to be most efficient, too. We also experimented with two-handed solutions and similar to the racing game we noticed the two-handed interaction to be significantly more exhausting. But on the contrary the subjects could handle the plane more accurately by using the steering wheel gesture. Because of that, switching between one and two-handed interaction was allowed at any time during play.

The most relevant digital controls in such a flight game are: primary fire (e.g. fire the gun), secondary fire (e.g. launch a rocket) and extend/retract undercarriage. After experimenting with several different combinations we came to the following solution: primary fire can be performed by moving the hand (only single-handed) to the right side of the working volume and secondary fire can be performed by a short and fast hand movement toward the left followed by a similar movement back to the right. This way, the duration of primary fire can easily be regulated and secondary fire can simultaneously be performed. Note that involuntary use of secondary fire induced by stopping primary fire is avoided by also conditioning secondary fire on the hand movement back to the right. For an example of one of the subjects using our interface see the accompanying video

and the image sequence depicted in Fig. 11.5.

Figure 11.5: An exemplary image sequence of playing Yager using our interface.

If both hands are tracked, the distance between the hands is used for firing instead of the X-coordinate of one hand; large hand distance for primary and short fast decrease and increase of the distance for secondary fire. The undercarriage is extended/retracted when the user forms the 'palm'-posture with one or both hands. This accounts for the higher demands on the fine motor skills for performing posture changes as this functionality is needed very infrequently.

This kind of interface turned out to be intuitive for the analog controls and easy-to-learn as well as easy-to-control for the digital mappings. The feedback from the subjects was very positive.

11.3.3 First-Person Shooter

The last game we have chosen is Quake II from ID Software released in 1997, a well known representative of the first-person shooter genre. Looking from the perspective of the player character, the user has to shoot enemies and make his way trough a futuristic alien environment. The game features items like weapons and power-ups which can be collected by moving the virtual player nearby. While power-ups are activated immediately, the player can switch between all the weapons already collected. The player has 4 continuous DOFs: moving forward/backward, shift left/right, turn left/right, look up/down. The most relevant digital controls are: fire, next/previous weapon, jump and crouch.

For this game we first concentrated on a single-hand mapping that should gather all the controls as intuitive and efficient as possible. Therefore the hand position is used for "walking" and "running". We map the controls such that the hand movement direction corresponds to the direction of the resulting in-game movement. The Z-coordinate is used for moving forward/backward and the X-coordinate for shifting left/right. The in-game speed of movement along an axis is determined by the hand's distance to the origin. The sizes of the according dead-zones strongly influenced the subjects' handling of the virtual player. If set to a small range (e.g. 2 cm in our final setting) an imminent change of movement is

easily possible with just a slight change in hand position. This allows fast control for an experienced player. However an unfamiliar user will experience problems finding a resting position. An enlarged deadzone would solve this problem but demands more spacious hand movements. Therefore, we use a large deadzone for unfamiliar users and allow for dynamically decreasing the deadzones for experienced users.

The hand orientation is used for "looking around". Note that this is also exploited for aiming at enemies, since weapons are always fired in direction of the screen center. The hand's pitch can be used directly as the absolute pitch for in-game view (scaled by 0.5 in our setting) for comfortable and precise control. Yaw however needs to be at least partly relative, since the virtual player has to be turned around horizontally in the full 360 degrees range, which is not applicable for the user hand. We tried several combinations of absolute and relative control, but unfortunately this always resulted in unintuitive handling. Therefore, we exploited a relative control with the non-linear mapping $f(x) = |x|x + \frac{x}{2}$, where x is the deflection of the yaw angle's analog control and the resulting value determines the horizontal rotational velocity in the game. This mapping allows for precise as well as fast turning and showed the best results in our experiments.

For the digital controls we used a left Roll Click for fire, a right Roll Click for next weapon, a lower bound on the Y-axis for crouch and a fast and small up-movement for jump. Since in Quake II all usable objects are activated instantly by touching (e.g. doors and buttons) we had not to address this functionality. For games where an activate functionality is needed we suggest to make use of the 'pointingB'-posture. For an example of one of the subjects using our interface see the accompanying video and the image sequence depicted in Fig. 11.6.

Figure 11.6: An exemplary image sequence of playing Quake II using our interface. Performed actions from left to right: turn right, change weapon, move straight, shoot, look up.

Since our first approach bundles all controls in one hand it turned out to be hard to handle for some subjects. Therefore, we also tried different two-handed configurations in order to distribute the functions on both hands. Unfortunately, when "moving" and "looking" were mapped on different hands the required asym-

metric coordination made control less intuitive for the subjects. Even after a decent time of testing the subjects did not show significant improvements. A more suitable setting is mapping the digital controls to one and the analog controls to the other hand. Like the one-hand setting, this was perceived as rather intuitive. Since it also bears the disadvantage of being more exhausting we integrated it as an optional approach to single-handed control.

The one-hand approach was found to be rather understandable and intuitive but also a bit too complex at the beginning. Problems occurred when subjects got into stressful in-game situations such as being simultaneously attacked by multiple enemies. The necessary movements like shifting aside and aiming at the enemies often resulted in a loss of orientation. Fortunately, the subjects adapted quickly and were in particular able to notably improve the accuracy of aiming after a short while. Nonetheless, we have to conclude that in terms of mere accuracy, our approach can not compete with a classic two-handed mouse/keyboard control. But considering the positive feedback of the subjects we think that bare-handed interaction might be similarly or even more enjoyable for this kind of game. Last but not least the one-handed approach is a possibility for disabled people not having the option of using both hands.

11.3.4 User Study

In order to gain feedback from potential end-users and evaluate the usability of our bare-hand gaming interfaces, we conducted a user study. It is based on game play in the three games Torcs (see Sec. 11.3.1), Yager (see Sec. 11.3.2) and Quake II (see Sec. 11.3.3).

In the following two subsections, we will outline this user study and present some of the results. A detailed description and discussion of the conducted study and its results can be found in [Zhe09].

Experimental Setting

In order to get comparable results, the hand-tracking virtual joystick device as well as a traditional game controller were tested. Since we developed the bare-handed interaction device as a universal controller for 3D computer games, it is not adequate to compare it with those devices being specially designed for a particular game genre, such as a force feedback steering wheel for racing games. Therefore, the Sony DualShock2 device was chosen. It provides analog controls using two sticks, a four-way digital cross (D-pad) and a set of action buttons including four shoulder buttons. The two analog sticks allow simultaneous control of 4 degrees of freedom. This device is designed for general use and can control various games using different mappings. For each game, the gamepad input was mapped to

different in-game controls in a standard way. The detailed mapping configurations can be seen in Table 11.1. In the interaction with the bare-handed device the final

Game	Game Control	Gamepad Control
Torcs	steering left/right	left stick left/right
	accelerate/reverse	right stick forward/backward
Yager	pitch	left stick forward/backward
	yaw	left stick left/right
	roll	right stick left/right
	increase throttle power	left shoulder button 1
	decrease throttle power	left shoulder button 2
	primary fire	right shoulder button 1
	secondary fire	right shoulder button 2
Quake II	sidestep left/right	left stick left/right
	move forward/backward	left stick forward/backward
	look up/down	right stick forward/backward
	turn left/right	right stick left/right
	jump	left shoulder button 1
	crouch	left shoulder button 2
	fire	right shoulder button 1
	next weapon	right shoulder button 2

Table 11.1: Control mapping configuration for the Sony DualShock2 gamepad.

mappings described in Sec. 11.3.1, 11.3.2 and 11.3.3 were used, respectively.

To find out people's personal views on and experiences of playing 3D games using only their bare hands, we designed a questionnaire. Designing questions in the questionnaire is dependent on what information we want to know. From our own experiences and the previous work of [HHTR05, PWS08, GFA09], we elicited the following factors as the most important issues for bare-handed gaming: intuitiveness, learnability, controllability, comfort, fun, and liking. These factors formulate a framework to analyze user preference in this context.

Our questionnaire consists of three sections and totally 35 questions. Section A focuses on different aspects of user preferences for the markerless hand-tracking interfaces. 13 statements were rated by the subjects for car racing, flight simulation and first-person shooting, respectively. Section B comprises 15 questions, which provided ratings for different game controllers (e.g. one hand, two hands, gamepad) in terms of fun, control accuracy, intuitiveness, learnability and comfort in all three games. In Section C, we asked 7 questions to collect the user's profile such as age, gender, dominant hand and playing skills. In Sections A and B, the typical 5-point likert rating scales [Lik32] were used for evaluating the

subject's attitude. The subjects chose a number from 1 to 5 using the following criteria: 1 - strongly disagree/very bad, 2 - disagree/bad, 3 - neutral/no opinion, 4 - agree/good, 5 - strongly agree/very good.

Additionally, in order to gain information about the efficiency of our interfaces, we also designed an objective testing scenario. User performance is measured by collecting completion times in predefined tasks. We used a video camera to capture the user's hand motion and the data on the screen simultaneously. Any aspects of the interaction can be analyzed in detail afterwards. Note that generally performing objective testing is very difficult and costly, because it needs high numbers of samples to counteract the impact of contingencies in the game. Moreover, it is time consuming to analyze the video data. In case of flight simulation and first-person shooting it is even more difficult, because both are mission-based games, so users do not win until the mission is accomplished. Furthermore, the in-game environment is so dynamic and complex that it is very hard to evaluate user performance. Therefore, the objective testing was only applied for the racing game Torcs. Each subject drove three laps of a specific track employing the hand-tracking device and the gamepad, respectively. Additionally, a car slalom task was designed. Subjects were asked to drive a car through 16 traffic cones, without missing or hitting a cone. For each controller each subject played the slalom 5 times. The completion times were obtained by analyzing the recorded video after the session.

15 people (3 female, 12 male) attended the user study. Most were between the ages of 17 and 30. Most were university students and came from different academic areas including agriculture, architecture, chemistry, computer science, economics, and mathematics. Considering game experience, racing games were the most regularly played games by our participants, followed by first-person shooter games. Flight simulator games were the most rarely played and about 67% of the participants had never played such a game. The objective testing was only performed with 10 of the 15 participants, because some had not enough time for this additional part.

Before the tests started, a brief introduction to our project was given and the subjects could familiarize with the interfaces and interaction devices. After that, each subject had about 5 minutes to play each game. Then, the performance testing of the racing game was carried out.

Results

Section A of the questionnaire focuses on various aspects of user preference for bare-handed gaming in different 3D games. In this context several statements had to be rated for each game. For lack of space and because these ratings exclude a comparison to another controller device, the results of Section A will only shortly

be summarized. In terms of control accuracy, comfort and recommendation the feedback was positive in average for each game, but also negative feedback was given from some subjects. For all games, in general the subjects gave a positive feedback in terms of intuitiveness, learnability and fun. Especially fun/excitement was rated very high (about 4.7) for each game. In summary, the feedback was quite encouraging for our research.

Section B of the questionnaire provides ratings for different controllers in terms of five factors that influence user preference: fun, control accuracy, intuitiveness, learnability and comfort. Thereby, ratings had to be given for interac-

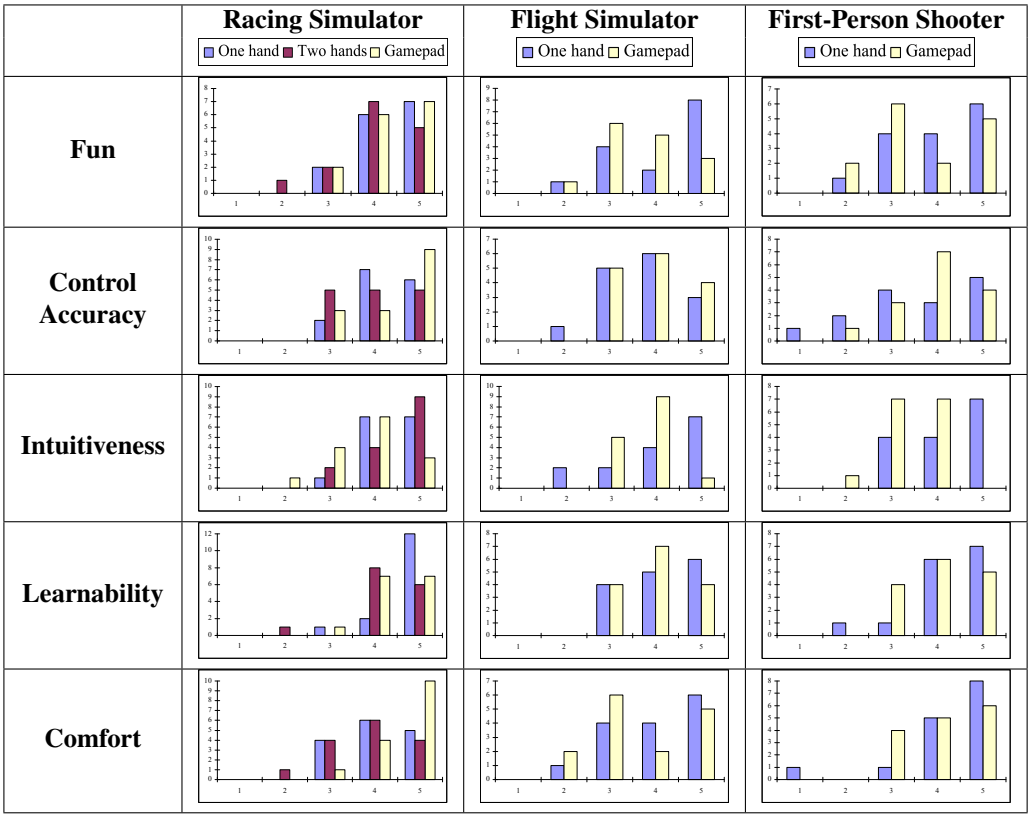


Table 11.2: Illustration of the results of Section B of the questionnaire. Each bar indicates the number of subjects (y-axes) that gave the respective rate (x-axes). Note that two-handed interaction was only tested for the racing simulator.

tion using both the bare hand(s) and a gamepad. A visual overview is given in Table 11.2. In terms of fun the differences between the controller devices were not significant except for the flight simulator, where the bare-handed control was rated clearly superior to the gamepad. Probably this is because the subjects were very unfamiliar with flight simulators and therefore could profit from the intuitiveness of bare-handed interaction. This argumentation is supported by the superior

results of bare-handed interaction in terms of intuitiveness and learnability. In these categories also the racing simulator and first-person shooter were rated significantly higher for bare-handed control. As expected, the control accuracy was rated inferior for bare-handed interaction, which is very likely due to the missing tactile feedback and practice. In terms of comfort the result was inconsistent for the different games. While for the racing simulator the gamepad was rated clearly superior, the rating for the other games was better for bare-handed control. We argue that this is because of the less intuitive handling using the gamepad in more complex tasks.

Altogether, the positive feedback from the questionnaires support our opinion that such a bare-handed interaction device can be a nice controller alternative for 3D games especially in terms of intuitiveness, learnability and fun.

In the objective testing, the mean lap time of each 3 laps of all 10 subjects was 39.91 seconds for using the bare hand and 41.54 seconds for the gamepad. No significant differences were found between the first, second and third lap timings, respectively. After performing the car slalom testing, the users' mean times of

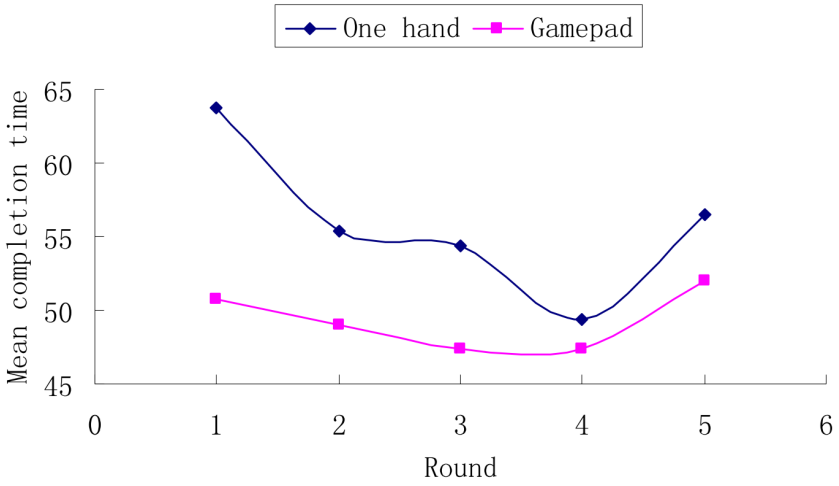


Figure 11.7: Mean completion times of the five rounds of the slalom task for either steering with one bare hand or the gamepad. A curve is quadratically fitted to the five discrete measuring points for illustration purposes.

5 rounds for finishing the task were calculated. We further computed the users' mean completion time for each round and both devices. The result (shown in Fig. 11.7) shows that after round-by-round playing, the subjects' mean completion time of the slalom task progressively decreased due to learning effects during the interaction. Moreover, the mean completion time with one hand decreases faster than with the gamepad. Surprisingly, in the fifth round the mean completion times for both devices increased again. We argue that this is because the subjects were

getting tired and lacked concentration after performing the same task for several times. We further argue that the generally better timings of the gamepad are due to the practice the subjects already had before the user study. The objective testing shows that bare-handed racing can also compete in terms of efficiency.

11.3.5 Interaction Design

Whenever hand motion is used for interaction, an important part is reducing fatigue. Moreover, a hand movement adopted for a specified action in-game should be intuitive or at least easy-to-learn. Based on our experimental tests several design principles for bare-handed interaction became apparent in this context:

- Minimize need of stretched arms in midair. Physical support (e.g. desk) or closer working space can help.
- Prefer rotational over translational degrees of freedom for movements that are needed very frequently, because for translations always the whole forearm has to be moved.
- Ensure that the mostly applied hand orientation is comfortable for the user. This is especially important for pitch and yaw, because bending the wrist joint is exhausting.
- Avoid high demands for the fine motor skills. For this reason, changing hand postures should only be used for infrequent actions.
- Avoid fast and spacious hand translations except exhaustion is part of the game (e.g. sport games).
- Give the opportunity to use only one hand. Using two hands is more exhausting than using only one hand.
- The directions of movement should correspond to the directions in the game (e.g. steering left should be performed by a left hand rotation or translation).
- Avoid exploiting too many DOFs if only little training time is available, because more DOFs need more familiarization time.
- Implement redundancies if possible. This better accounts for different user preferences.

These principles might support other developers to design suitable interaction metaphors for gaming.

CHAPTER 12

RELATED AND CONCURRENT WORK

The different types of interfaces to augment user interaction in Mixed Reality interfaces can be classified into three different categories (according to Wanderley et al. [WKCT06]):

1. Tangible interfaces: physical objects (e.g. a cube, a tool, etc.) are exploited to control manipulations of a virtual counterpart. For example, in the game mulTetris [ABCD06] the bricks of the traditional tetris game could be manipulated by real brick-like devices.

2. Traditional VR interfaces: physical controller devices (e.g. gloves, joysticks, wands, infrared sensors, etc.) are used for user interaction. For example in the game ARQuake [TCD+02] augmented reality technology (e.g. head mounted displays, GPS-tracked laptops) is massively used for interaction. Also the Nintendo Wii-Remote belongs to this category.

3. Bare-handed interfaces: the user body (or parts of it) is exploited as an interaction device. No markers or physical devices are needed. This way, more natural interaction can be designed (according to Winkler et al. [WYZ07]).

This work presents novel bare-handed interaction interfaces for different purposes. Therefore, in the following we will only focus on literature in this particular field. Readers being interested in other approaches we refer to [vHB01] and [BKLP05]. Von Hardenberg and Bérard [vHB01] provided a brief overview of earlier bare-handed human-computer interaction. Bowman et al. [BKLP05] provide an extensive overview of 3D interaction interfaces and techniques.

12.1 Serious Applications

One of the most frequently investigated application of bare-handed control is emulating the standard computer mouse. Examples are the Fingertrack system of O'Hagan and Zelinsky [OZ97] and the pinch gesture based cursor control of Wil-

son [Wil06]. In both systems the cursor is moved according to the hand translation and postures are exploited for clicking purposes.

Similar to mouse emulation is designing a bare-handed drawing application as for instance done in [MI00, LL01, ASO00]. Thereby, different postures are used to indicate whether the pointer should draw or not. Hand translation is used to move the virtual paint-brush.

Other mouse related applications are bare-handed television control [FAB+98] and bare-handed control of music software [KCF09]. In [FAB+98] using a trigger gesture the television control is enabled. Several settings (e.g. volume) can be adjusted as with a mouse by hand motion. In [KCF09] hand movements can be used to control granular synthesis parameters in a music application and in a sound mixing scenario virtual sliders can be controlled with a hand.

Another oftentimes investigated application is 2D and 3D virtual object manipulation. In 2D mostly the bare-handed manipulation of images was approached as for instance in [WYZ07] and [DMR06]. Using hand motion and different postures images could be inspected and simple manipulation operations such as zoom/resize could be performed. An example for 3D manipulation with bare hands is [SSK01]. Changing the hand posture is used to switch the current mode such that a 3D model can be translated, rotated and scaled.

For navigating in terrain data two different typical approaches exist. The first solution is designing a 2D control as done in [Wil06]. The terrain data can be explored from above by panning, rotating in the plane of the keyboard and zooming. Thereby both a one-handed and a two-handed technique were introduced. The second solution for the navigation in terrain data is using the hand motion to control a fly through as done in [OZR02] by adopting the flying vehicle control metaphor [WO90] (see Sec. 7.5).

While the mentioned approaches provide nice opportunities for bare-handed interface design, most of them suffer from employing restricted hand-tracking systems. None of the used systems were capable of tracking the full global hand pose without restrictions. Most of the presented interfaces are designed to optimize interaction with very specific tracking dependent preconditions. However, whether the specific kind of interaction in general a good solution remains unclear.

12.2 3D Games

Freeman et al. [FTOK96] developed hardware and algorithms for a bare-hand-tracking system to fulfill the high speed and low-cost requirement for computer games. Four hand parameters could be recognized this way: the two coordinates of the position in the camera image, the orientation parallel to the image plane and the width of the hand. These parameters were exploited to control a racing car in

a simple way. Unfortunately, the interface design was strongly restricted by the little amount of available DOFs.

Segen and Kumar [SK98] introduced the 3D hand interaction interface GestureVR. Two cameras were used to capture the players' hand motion, and three postures were defined by tracking the thumb and the index finger in real-time (60Hz). For each finger, five spatial parameters, consisting of finger tip positions (X, Y, Z), azimuth angle and elevation angle of the finger axis, were calculated. Two applications were developed based on this system: a fly-through application and a first-person shooter (FPS). In the fly-through application the user can control the flight of a virtual camera based on the index fingertip position and the hand's pitch, yaw and roll angles. For the FPS only one camera is used for tracking the hand. Because of that, less DOFs were available for interaction. Only forward/reverse speed, lateral motion and the angular velocity of turning the player left/right were controlled by the hand pose. Door opening and weapon fire were mapped to different hand postures. However, current FPS need further controls to be handled (e.g. look up/down).

For a FPS game Kang et al. [KLJ04] were the first to exploit bare-handed upper-body gestures for game control. 10 predefined gestures were mapped to specific controls in the game Quake II. Park et al. [PJK06] improved this interface mostly with respect to the requirements on the tracking environment. Though such an interface enables users to play with natural gestures, it is quite exhausting to use full body motion especially in such a game scenario, because a FPS requires fast and frequent actions.

Lu et al. [LCZW05] presented a vision-based fist-tracking method to control their own racing application. Both hand positions are tracked and a steering wheel gesture (angle of the line connecting both hand positions) is used for steering left and right. However, this approach was very simple and the hand orientation was not taken into account.

Recently, Song et al. [SYW08] introduced a vision-based 3D finger interaction approach to control two self-implemented games: a finger fishing game, where small virtual objects have to be angled with the index finger, and a virtual version of Jenga, where parts of a tower of wooden blocks have to be extracted successively while trying not to topple the tower. These games are nice examples of how traditional games with physical interaction can be implemented in Virtual Reality using vision and physics simulation. But while Song et al. [SYW08] designed novel games for bare-handed interaction, the focus of our work is designing suitable bare-handed interaction interfaces for existing standard games.

CONCLUSIONS

Within the scope of this thesis a novel real-time bare-hands-tracking device was developed. On the basis of this device a comprehensive investigation of bare-handed human computer interaction was performed. In this context, several novel interaction techniques and interfaces were developed solving general drawbacks as well as better exploiting the capabilities of this novel device. This thesis deposits a basis for bare-handed human computer interaction. We argue that this is the first time a bare-hand-tracking device capable of real-time tracking the full global hand poses of both human hands is implemented and in depth investigated for its use in various interaction scenarios. In the following, the respective parts of this thesis are concluded in more detail.

Hand-Tracking

For the simultaneous real-time tracking of both human hand poses a novel method was introduced. This method is additionally capable of recognizing and distinguishing 4 different stiff postures per hand. To this end, first, a procedure to track one hand from the segmented images of three cameras was developed. In contrast to existing approaches, which typically first analyze the segmented 2D images separately and then aim at discovering the hand pose based on extracted 2D image features, we first combine the segmented 2D images to one coarse 3D representation of the hand by using the shape from silhouettes technique. Then, 3D fingertip features are detected and identified by analyzing the 3D representation. This way, a more robust system could be constructed which not longer suffer from angle limitations of the tracked hand pose.

In a next step, this one-hand-tracking procedure was extended for enabling the tracking of both human hands simultaneously. In this context we solved problems such as distinguishing the segmented hand masks into disjunct sets each belonging to one hand and discovering for a tracked hand whether it is the left or the right hand.

Furthermore, we presented a novel method for automatic analysis and control of the image segmentations. Therefore, the image segmentations are analyzed in real-time and depending on the result the segmentation can be improved or the user can be instructed to use the device correctly. This way, the hand-tracking

system can run over a long period of time without supervision by an administrator. This method is not restricted to the employed segmentation technique in our system; also other segmentation methods can profit from its use.

Interaction Techniques

With a novel interaction device having specific properties and requirements typically existing interaction techniques have to be adapted or novel interaction techniques have to be developed in order to fully exploit the capabilities or compensate the drawbacks of this device. Therefore, several novel interaction techniques as well as extensions of existing interaction techniques were introduced. However, these techniques are not restricted to our device, also other interaction devices can profit from their use. Several techniques were introduced to improve the following fundamental mechanisms:

Selection: To improve clicking simulation for selection purposes the Roll Click gesture was introduced, which enables efficient and easy clicking with a small hand twist around the roll axis. Furthermore, for solving the problem of imprecise cursor movements due to natural hand tremor or hand-tracking inaccuracies the Hybrid Cursor Control was introduced. With this technique cursor positioning can be performed more precisely while the intuitiveness is further maintained. Its superior applicability was confirmed in a user experiment.

Manipulation: To solve the problem of precisely grabbing and releasing 3D objects in virtual environments the Jerky Release technique was developed. Furthermore, its applicability was investigated by an informal user study with a superior result in complex manipulation tasks. In the context of two-handed manipulation of 3D objects, the Bi-manual Symmetric Grab technique was introduced, which enables efficient grabbing, moving and releasing virtual objects by using both hands. A user experiment showed the superior applicability compared to several recent approaches in particular for high precision tasks.

Travel: To travel in a virtual environment the steering technique was adapted to the needs of bare-handed interaction.

Feedback: For at least partly compensating the missing feedback in bare-handed interaction, the Visual Feedback Box was introduced, which particularly supports the familiarization with the new device and different interaction techniques.

Having addressed these fundamental issues, bare-handed interaction interfaces can be built on top of such a device and really exploit its capabilities.

Interaction Interfaces

The last step for bringing bare-handed human computer interaction into practice is designing interaction interfaces. Suitable interaction techniques have to be selected and combined to meaningful interaction metaphors dependent on the device and the specific requirements of an application. In this context we introduced interfaces for several self-implemented as well as commercial applications.

A simple virtual building blocks application and a mesh editing application were implemented and interfaced to the hand-tracking device using standard interaction techniques. Although these applications could be handled by most users, some problems of bare-handed interaction became apparent such as the absence of physical buttons and the missing feedback from the hand-tracking device. Therefore, we designed a more complex 3D interface enabling fundamental manipulations of 3D objects, which makes use of our novel interaction techniques. This way, more efficient interaction could be designed. This interface also served as an experimental platform for testing novel interaction techniques.

To generally allow for controlling various applications via bare-hands we developed generic interfaces to simulate the 2D mouse as well as a joystick device. Therefore, our novel interaction techniques for selection were combined to build a pointing device enabling 2D mouse simulation. The virtual joystick device is simulated by building upon a virtual joystick driver. This joystick device can be employed and configured for every application having joystick support. Four commercial 3D games belonging to different typical game genres were connected to this device. By performing several informal experiments suitable interaction metaphors could be investigated and obtained. In a more formal user study the identified interaction metaphors are then further evaluated and compared to a traditional controller device. As a result from the experiments several general design principles were identified supporting other developers to design reasonable bare-handed interaction for games.

Last but not least two prevalent commercial applications were interfaced to the hand-tracking device: a fly-through interface for controlling Zugspitze 3D and a 3D manipulation interface for handling RTT DeltaGen. These interfaces make use of several interaction techniques introduced in this thesis.

Future Directions

Future work might strive for fewer requirements on the environment of the hand-tracking system in order to use the device in more general settings. This would

require additional techniques for segmenting the hands in the camera images. We think that in the near future depth cameras with sufficient resolutions and frame rates will be available that solve this problem. Then, beside segmentation the depth information could also be exploited to improve the 3D reconstruction of the hand. Probably, with this information even markerless tracking all DOFs of the human hands could robustly be possible in real-time including automatic initialization.

As soon as a hand-tracking system capable of tracking more DOFs than ours is available, also novel interaction techniques and interfaces will be needed to fully exploit its capabilities. But even for a hand-tracking device like ours, we think that research and investigation on interaction techniques is still needed. Especially, for bi-manual asymmetric interaction further work could be performed.

In consideration of the large amount of different existing applications and the ongoing development of new applications the need for designing suitable interaction interfaces will also persist. Furthermore, with novel interaction devices such as ours, also novel applications and games could and should be developed, which are better tailored to the particular needs and qualities of the specific interaction device.

BIBLIOGRAPHY

[ABCD06] S. Audet, M. Bedrosian, C. Clement, and M. Dinculescu. Multetris: A test of graspable user interfaces in collaborative games, 2006.

[AC99] J. K. Aggarwal and Q. Cai. Human motion analysis: A review. *Computer Vision and Image Understanding*, 73:428–440, 1999.

[ASO00] K. Abe, H. Saito, and S. Ozawa. 3-d drawing system via hand motion recognition from two cameras. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 2, pages 840–845 vol.2, 2000.

[Bal04] Ravin Balakrishnan. "beating" fitts' law: virtual enhancements for pointing facilitation. *Int. J. Hum.-Comput. Stud.*, 61(6):857–874, 2004.

[BBM+06] Brian Badillo, Doug A. Bowman, William McConnel, Tao Ni, and Mara G. da Silva. Literature survey on interaction techniques for large displays, 2006.

[BCR+03] Patrick Baudisch, Edward Cutrell, Dan Robbins, Mary Czerwinski, Peter Tandler, Peter T, Benjamin Bederson, and Alex Zierlinger. Drag-and-pop and drag-and-pick: techniques for accessing remote screen content on touch- and pen-operated systems. In *Proceedings of Interact 2003*, pages 57–64, 2003.

[BEM07] P. Breuer, C. Eckes, and S. Muller. Hand gesture recognition with a novel ir time-of-flight range camera: A pilot study. In *MIRAGE07*, pages 247–260, 2007.

[BGBL04] Renaud Blanch, Yves Guiard, and Michel Beaudouin-Lafon. Semantic pointing: improving target acquisition with control-display ratio adaptation. In *CHI*, pages 519–526, 2004.

[BH97] Doug A. Bowman and Larry F. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive

virtual environments. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 35–38, New York, NY, USA, 1997. ACM.

[BI05] Christoph W. Borst and Arun P. Indugula. Realistic virtual grasping. In *VR '05: Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, pages 91–98, Washington, DC, USA, 2005. IEEE Computer Society.

[BKLP05] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola, and Ivan Poupyrev. *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2005.

[BKMSG04] Matthieu Bray, Esther Koller-Meier, Nicol N. Schraudolph, and Luc Van Gool. Stochastic meta-descent for tracking articulated structures. In *IEEE Workshop on Articulated and Nonrigid Motion, Conference on Computer Vision and Pattern Recognition (CVPR)*, Washington, D.C., 2004.

[Bol80] Richard A. Bolt. "put-that-there": Voice and gesture at the graphics interface. *SIGGRAPH Comput. Graph.*, 14(3):262–270, 1980.

[BPS+09] Kelly Byron, Linh Pham, Joshua Situka, Ken Kopecky, and Song Zhang. Markerless motion tracking. Graduate Program of Human Computer Interactionm, Virtual Reality Applications Center, Iowa State University, 2009.

[CB06] Andy Cockburn and Philip Brock. Human on-line response to visual and motor target expansion. In *GI '06: Proceedings of Graphics Interface 2006*, pages 81–87, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[CBC95] James L. Crowley, François Bérard, and Joëlle Coutaz. Finger tracking as an input device for augmented reality, 1995.

[CFH97] Lawrence D. Cutler, Bernd Fröhlich, and Pat Hanrahan. Two-handed direct manipulation on the responsive workbench. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 107–114, New York, NY, USA, 1997. ACM Press.

[CH96] Roberto Cipolla and Nicholas J. Hollinghurst. Human-robot interface by pointing with uncalibrated stereo vision. *Image and Vision Computing*, 14(3):171 – 178, 1996.

[Che04] Shinko Y. Cheng. Hand pose estimation using expectation-constrained-maximization from voxel data. Technical report, University of California, San Diego, 2004.

[CHWS88] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman. An empirical comparison of pie vs. linear menus. In *CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 95–100, New York, NY, USA, 1988. ACM.

[CK03] Gilbert Cockton and Panu Korhonen, editors. *Proceedings of the 2003 Conference on Human Factors in Computing Systems, CHI 2003, Ft. Lauderdale, Florida, USA, April 5-10, 2003*. ACM, 2003.

[CUK+08] A. Causo, E. Ueda, Y. Kurita, Y. Matsumoto, and T. Ogasawara. Model-based hand pose estimation using multiple viewpoint silhouette images and unscented kalman filter. In *Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on*, pages 291–296, Aug. 2008.

[dLGP06] M. de La Gorce and N. Paragios. Monocular hand pose estimation using variable metric gradient-descent. In *BMVC06*, page III:1269, 2006.

[dLGPF08] M. de La Gorce, N. Paragios, and D.J. Fleet. Model-based hand tracking with texture, shading and self-occlusions. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[DMR06] Pushkar Dhawale, Masood Masoodian, and Bill Rogers. Bare-hand 3d gesture input to interactive systems. In *CHINZ '06: Proceedings of the 7th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction*, pages 25–32, New York, NY, USA, 2006. ACM.

[DS99] J. Davis and M. Shah. Toward 3-d gesture recognition. *PRAI*, 13(3):381, May 1999.

[EBN+05] Ali Erol, George Bebis, Mircea Nicolescu, Richard D. Boyle, and Xander Twombly. A review on vision-based full dof hand motion estimation. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, page 75, Washington, DC, USA, 2005. IEEE Computer Society.

[EBN+07] Ali Erol, George Bebis, Mircea Nicolescu, Richard D. Boyle, and Xander Twombly. Vision-based hand pose estimation: A review. *Comput. Vis. Image Underst.*, 108(1-2):52–73, 2007.

[FAB+98] William T. Freeman, David B. Anderson, Paul A. Beardsley, Chris N. Dodge, Michal Roth, Craig D. Weissman, William S. Yerazunis, Hiroshi Kage, Kazuo Kyuma, Yasunari Miyake, and Ken ichi Tanaka. Computer vision for interactive computer graphics. *IEEE Computer Graphics and Applications*, 18(3):42–53, 1998.

[FK05] Scott Frees and G. Drew Kessler. Precise and rapid interaction through scaled manipulation in immersive virtual environments. In *VR '05: Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, pages 99–106, Washington, DC, USA, 2005. IEEE Computer Society.

[FKK07] Scott Frees, G. Drew Kessler, and Edwin Kay. Prism interaction for enhancing control in immersive virtual environments. *ACM Trans. Comput.-Hum. Interact.*, 14(1):2, 2007.

[FTOK96] W.T. Freeman, K. Tanaka, J. Ohta, and K. Kyuma. Computer vision for computer games. In *Proceedings of the Second International Conference on Automatic Face and Gesture Recognition*, pages 100–105, Oct 1996.

[Gav99] D. M. Gavrila. The visual analysis of human movement: a survey. *Comput. Vis. Image Underst.*, 73(1):82–98, 1999.

[GAW+06] T. Gumpp, P. Azad, K. Welke, E. Oztop, R. Dillmann, and G. Cheng. Unconstrained real-time markerless hand tracking for humanoid interaction. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 88–93, Dec. 2006.

[GBBL04] Yves Guiard, Renaud Blanch, and Michel Beaudouin-Lafon. Object pointing: a complement to bitmap pointing in guis. In *GI '04: Proceedings of Graphics Interface 2004*, pages 9–16, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[GBCB00] R. Grzeszcuk, G. Bradski, M.H. Chu, and J.-Y. Bouguet. Stereo based gesture recognition invariant to 3d pose and lighting. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 826–833 vol.1, 2000.

[GFA09] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. A survey of software learnability: metrics, methodologies and guidelines. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 649–658, New York, NY, USA, 2009. ACM.

[GFGB04] C. Grätzel, T. Fong, S. Grange, and C. Baur. A non-contact mouse for surgeon-computer interaction. *Technol. Health Care*, 12(3):245–257, 2004.

[GLA+08] Seyed Eghbal Ghobadi, Omar Edmond Loepprich, Farid Ahmadov, Jens Bernshausen, Klaus Hartmann, and Otmar Loffeld. Real time hand based robot control using 2d/3d images. In *ISVC '08: Proceedings of the 4th International Symposium on Advances in Visual Computing, Part II*, pages 307–316, Berlin, Heidelberg, 2008. Springer-Verlag.

[Gui87] Yves Guiard. Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *Journal of Motor Behavior*, 19(4):486–517, 1987.

[GWB05] Tovi Grossman, Daniel Wigdor, and Ravin Balakrishnan. Multi-finger gestural interaction with 3d volumetric displays. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 931–931, New York, NY, USA, 2005. ACM.

[Hec06] D. Heckenberg. Performance evaluation of vision-based high dof human movement tracking: A survey and human computer interaction perspective. In *V4HCI06*, page 156, 2006.

[HHTR05] Johanna Höysniemi, Perttu Hämäläinen, Laura Turkki, and Teppo Rouvi. Children's intuitive gestures in vision-based action games. *Commun. ACM*, 48(1):44–50, 2005.

[HMF08] M. B. Holte, T. B. Moeslund, and P. Fihl. View invariant gesture recognition using the csem swissranger sr-2 camera. *Int. J. Intell. Syst. Technol. Appl.*, 5(3/4):295–303, 2008.

[Jen99] Cullen Jennings. Robust finger tracking with multiple cameras. In *RATFG-RTS '99: Proceedings of the International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, page 152, Washington, DC, USA, 1999. IEEE Computer Society.

[JKS98] K.-H. Jo, Y. Kuno, and Y. Shirai. Manipulative hand gesture recognition using task knowledge for human computer interaction. *Automatic Face and Gesture Recognition, IEEE International Conference on*, 0:468, 1998.

[JS07] Alejandro Jaimes and Nicu Sebe. Multimodal human-computer interaction: A survey. *Comput. Vis. Image Underst.*, 108(1-2):116–134, 2007.

[KB95] Paul Kabbash and William Buxton. The "prince" technique: Fitts' law and selection using area cursors. In *CHI*, pages 273–279, 1995.

[KCF09] Chris Kiefer, Nick Collins, and Geraldine Fitzpatrick. Phalanger: Controlling music software with hand movement using a computer vision and machine learning approach. In *New Interfaces For Musical Expression*, Pittsburgh, USA, June 2009.

[KHM+98] James T. Klosowski, Martin Held, Joseph S. B. Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[KHW95] G. Drew Kessler, Larry F. Hodges, and Neff Walker. Evaluation of the CyberGlove as a whole-hand input device. *ACM Transactions on Computer-Human Interaction*, 2(4):263–283, 1995.

[KLJ04] Hyun Kang, Chang Woo Lee, and Keechul Jung. Recognition-based gesture spotting in video games. *Pattern Recogn. Lett.*, 25(15):1701–1714, 2004.

[KLK07] F. Kahlesz, C. Lilge, and R. Klein. Easy-to-use calibration of multiple-camera setups. *CCMVS: Camera Calibration Methods for Computer Vision Systems*, 2007.

[KQL06] Aaron Kotranza, John Quarles, and Benjamin Lok. Mixed reality: are two hands better than one? In *VRST '06: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 31–34, New York, NY, USA, 2006. ACM Press.

[KSK01] Hideki Koike, Yoichi Sato, and Yoshinori Kobayashi. Integrating paper and digital information on enhanceddesk: a method for real-time finger tracking on an augmented desk system. *ACM Trans. Comput.-Hum. Interact.*, 8(4):307–322, 2001.

[KSMB08] Regis Kopper, Mara G. Silva, Ryan P. McMahan, and Doug A. Bowman. Increasing the precision of distant pointing for large high-resolution displays, 2008.

[KT04a] Mathias Kölsch and Matthew Turk. Robust hand detection. In *In International Conference on Automatic Face and Gesture Recognition (to appear), Seoul, Korea*, pages 614–619, 2004.

[KT04b] Mathias Kolsch and Matthew Turk. Analysis of rotational robustness of hand detection with a viola-jones detector. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3*, pages 107–110, Washington, DC, USA, 2004. IEEE Computer Society.

[Lau94] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(2):150–162, 1994.

[LB04] Julien Letessier and François Bérard. Visual tracking of bare fingers for interactive surfaces. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 119–122, New York, NY, USA, 2004. ACM.

[LCZW05] P. Lu, Y.F. Chen, X.Y. Zeng, and Y.S. Wang. A vision based game control method. In *CVHCI05*, page 70, 2005.

[Lik32] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55, 1932.

[LKG+03] Ignacio Llamas, Byungmoon Kim, Joshua Gargus, Jarek Rossignac, and Chris D. Shaw. Twister: a space-warp operator for the two-handed editing of 3d shapes. *ACM Trans. Graph.*, 22(3):663–668, 2003.

[LL01] Ivan Laptev and Tony Lindeberg. Tracking of multi-state hand models using particle filtering and a hierarchy of multi-scale image features. In *Scale-Space '01: Proceedings of the Third International Conference on Scale-Space and Morphology in Computer Vision*, pages 63–74, London, UK, 2001. Springer-Verlag.

[LMS03] M. Li, M. Magnor, and H.-P. Seidel. Hardware-accelerated visual hull reconstruction and rendering. In *Proc. Graphics Interface (GI'03),* Halifax, Canada, pages 65–71, June 2003.

[LTCK03] Alan Liu, Frank Tendick, Kevin Cleary, and Christoph Kaufmann. A survey of surgical simulation: applications, technology, and education. *Presence: Teleoper. Virtual Environ.*, 12(6):599–614, 2003.

[MA07] S. Mitra and T. Acharya. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3):311–324, May 2007.

[MB05] Michael J. McGuffin and Ravin Balakrishnan. Fitts' law and expanding targets: Experimental studies and designs for user interfaces. *ACM Trans. Comput.-Hum. Interact.*, 12(4):388–422, 2005.

[MF04] N. Murray and T. Fernando. An immersive assembly and maintenance simulation environment. In *DS-RT '04: Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 159–166, Washington, DC, USA, 2004. IEEE Computer Society.

[MFPBS97] Mark R. Mine, Jr. Frederick P. Brooks, and Carlo H. Sequin. Moving objects in space: exploiting proprioception in virtual-environment interaction. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 19–26, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[MG01] Thomas B. Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Comput. Vis. Image Underst.*, 81(3):231–268, 2001.

[MI00] John MacCormick and Michael Isard. Partitioned sampling, articulated objects, and interface-quality hand tracking. In *ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part II*, pages 3–19, London, UK, 2000. Springer-Verlag.

[ML04] Shahzad Malik and Joe Laszlo. Visual touchpad: a two-handed gestural input device. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces*, pages 289–296, New York, NY, USA, 2004. ACM Press.

[MLN05] Zhenyao Mo, J. P. Lewis, and Ulrich Neumann. Smartcanvas: a gesture-driven intelligent drawing desk system. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 239–243, New York, NY, USA, 2005. ACM.

[MM95] D. Mapes and J. M. Moshell. A two-handed interface for object manipulation in virtual environments. *Presence: Teleoperators and Virtual Environments*, 4(4):403–416, 1995.

[MS08] S. Malassiotis and M. G. Strintzis. Real-time hand posture recognition using range data. *Image Vision Comput.*, 26(7):1027–1037, 2008.

[MZ07] Daniel Mohr and Gabriel Zachmann. Segmentation of distinct homogeneous color regions in images. In Martin Kampel, editor, *The 12th International Conference on Computer Analysis of Images and Patterns (CAIP)*, pages 432–440, Vienna, Austria, 27–29 August 2007. Springer Verlag.

[MZ09] Daniel Mohr and Gabriel Zachmann. Continuous edge gradient-based template matching for articulated objects. In Hélder J. Araújo, editor, *International Conference on Computer Vision Theory and Applications (VISAPP)*, pages 519–524, Lisbon, Portugal, 05–08 February 2009. Insticc Press.

[OB04] Eng-Jon Ong and R. Bowden. A boosted classifier tree for hand shape detection. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 889–894, May 2004.

[OBL+05] T. Oggier, B. B¨uttgen, F. Lustenberger, G. Becker, B. R¨uegg, and A. Hodac. Swissranger sr3000 and first experiences based on miniaturized 3d-tof cameras. In *In Proc. of the First Range Imaging Research Day at ETH Zurich*, 2005.

[OH99] H. Ouhaddi and P. Horain. Hand tracking by 3d model registration, 1999.

[OKF+05] Russell Owen, Gordon Kurtenbach, George Fitzmaurice, Thomas Baudel, and Bill Buxton. When it gets more difficult, use both hands: exploring bimanual curve manipulation. In *GI '05: Proceedings of Graphics Interface 2005*, pages 17–24, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005. Canadian Human-Computer Communications Society.

[Osa06] Noritaka Osawa. Automatic adjustments for efficient and precise positioning and release of virtual objects. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality*

continuum and its applications, pages 121–128, New York, NY, USA, 2006. ACM.

[OSK02] Kenji Oka, Yoichi Sato, and Hideki Koike. Real-time tracking of multiple fingertips and gesture recognition for augmented desk interface systems. *Automatic Face and Gesture Recognition, IEEE International Conference on*, 0:0429, 2002.

[OZ97] Rochelle O'Hagan and Alexander Zelinsky. Finger track - a robust and real-time gesture interface. In *AI '97: Proceedings of the 10th Australian Joint Conference on Artificial Intelligence*, pages 475–484, London, UK, 1997. Springer-Verlag.

[OZR02] R. G. O'Hagan, A. Zelinsky, and S. Rougeaux. Visual gesture interfaces for virtual environments. *Interacting with Computers*, 14(3):231 – 250, 2002.

[PBWI96] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The go-go interaction technique: non-linear mapping for direct manipulation in vr. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 79–80, New York, NY, USA, 1996. ACM.

[PDK07] Nikolas Paries, Patrick Degener, and Reinhard Klein. Simple and efficient mesh editing with consistent local frames. In *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 461–464, Washington, DC, USA, 2007. IEEE Computer Society.

[PIWB98] I. Poupyrev, T. Ichikawa, S. Weghorst, and M. Billinghurst. Egocentric object manipulation in virtual environments: Empirical evaluation of interaction techniques. *Computer Graphics Forum*, 17(3):41–52, 1998.

[PJK06] H.S. Park, D.J. Jung, and H.J. Kim. Vision-based game interface using human gesture. In *PSIVT06*, pages 662–671, 2006.

[Pop07] Ronald Poppe. Vision-based human motion analysis: An overview. *Computer Vision and Image Understanding*, 108(1-2):4 – 18, 2007. Special Issue on Vision for Human-Computer Interaction.

[PSP99] Jeffrey S. Pierce, Brian C. Stearns, and Randy Pausch. Voodoo dolls: seamless interaction at multiple scales in virtual environments. In *I3D '99: Proceedings of the 1999 symposium on In-*

teractive 3D graphics, pages 141–145, New York, NY, USA, 1999. ACM.

[PWS08] David Pinelle, Nelson Wong, and Tadeusz Stach. Heuristic evaluation for games: usability principles for video game design. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1453–1462, New York, NY, USA, 2008. ACM.

[RK95] J. M. Rehg and T. Kanade. Model-based tracking of self-occluding articulated objects. In *ICCV '95: Proceedings of the Fifth International Conference on Computer Vision*, page 612, Washington, DC, USA, 1995. IEEE Computer Society.

[SBK09] Markus Schlattmann, Johannes Broekelschen, and Reinhard Klein. Real-time bare-hands-tracking for 3d games. In *IADIS International Conference Game and Entertainment Technologies (GET '09)*, pages 59–66. IADIS Press, June 2009.

[SCP95] Richard Stoakley, Matthew J. Conway, and Randy Pausch. Virtual reality on a wim: interactive worlds in miniature. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 265–272, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[SK98] Jakub Segen and Senthil Kumar. Gesture vr: vision-based 3d hand interace for spatial interaction. In *MULTIMEDIA '98: Proceedings of the sixth ACM international conference on Multimedia*, pages 455–464, New York, NY, USA, 1998. ACM.

[SK99] J. Segen and S. Kumar. Shadow gestures: 3d hand pose estimation using a single camera. In *CVPR99*, pages I: 479–485, 1999.

[SK07] Markus Schlattmann and Reinhard Klein. Simultaneous 4 gestures 6 dof real-time two-hand tracking without any markers. In *ACM Symposium on Virtual Reality Software and Technology (VRST '07)*, pages 39–42, November 2007.

[SK09a] Markus Schlattmann and Reinhard Klein. Efficient bimanual symmetric 3d manipulation for markerless hand-tracking. In *Virtual Reality International Conference (VRIC)*, April 2009.

[SK09b] Markus Schlattmann and Reinhard Klein. Hybrid cursor control, December 2009. EP09015378.4, European Patent Application.

[SK09c] Markus Schlattmann and Reinhard Klein. Hybrid cursor control for precise and fast positioning without clutching. In *SIGGRAPH ASIA '09: ACM SIGGRAPH ASIA 2009 Sketches*, page 1, New York, NY, USA, December 2009. ACM.

[SK09d] Markus Schlattmann and Reinhard Klein. Method for automatically detecting at least the type and/or location of a gesture formed using an appendage, particularly a hand gesture, March 2009. Rheinische Friedrich-Wilhelms Universität Bonn, WO 2009/027307 A1, Pr.: DE102007041482A1 31.08.2007, International Patent.

[SK10] Markus Schlattmann and Reinhard Klein. Efficient bimanual symmetric 3d manipulation for bare-handed interaction. *Accepted for publication in Journal of Virtual Reality and Broadcasting (JVRB)*, 2010.

[SKSK07] Markus Schlattmann, Ferenc Kahlesz, Ralf Sarlette, and Reinhard Klein. Markerless 4 gestures 6 dof real-time visual tracking of the human hand with automatic initialization. *Computer Graphics Forum*, 26(3):467–476, September 2007.

[SMP05] Tomás Svoboda, Daniel Martinec, and Tomás Pajdla. A convenient multicamera self-calibration for virtual environments. *Presence: Teleoper. Virtual Environ.*, 14(4):407–422, 2005.

[SNNK09] Markus Schlattmann, Tanin Na Nakorn, and Reinhard Klein. 3d interaction techniques for 6 dof markerless hand-tracking. In *International Conference on Computer Graphics, Visualization and Computer Vision (WSCG '09)*, February 2009.

[SPHK08] Stefan Soutschek, Jochen Penne, Joachim Hornegger, and Johannes Kornhuber. 3-D Gesture-Based Scene Navigation in Medical Imaging Applications Using Time-Of-Flight Cameras. In Omnipress IEEE Computer Society Conference on Computer Vision, editor, *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2008.

[SSK01] Yoichi Sato, Makiko Saito, and Hideki Koik. Real-time input of 3d pose and gestures of a user's hand and its applications for hci. In *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)*, page 79, Washington, DC, USA, 2001. IEEE Computer Society.

[Ste04] B. Stenger. *Model-Based Hand Tracking Using A Hierarchical Bayesian Filter*. PhD thesis, University of Cambridge, Cambridge, UK, March 2004.

[Stu92] D. J. Sturman. *Whole-Hand Input*. PhD thesis, MIT, 1992.

[SYW08] Peng Song, Hang Yu, and Stefan Winkler. Vision-based 3d finger interactions for mixed reality games with physics simulation. In *VRCAI '08: Proceedings of The 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, pages 1–6, New York, NY, USA, 2008. ACM.

[SZBK10] Markus Schlattmann, Tan Zheng, Johannes Broekelschen, and Reinhard Klein. An investigation of bare-hands-interaction in traditional 3d game genres. *Accepted for IADIS International Journal on WWW/Internet (IJWI)*, 2010.

[TCD+02] Bruce Thomas, Ben Close, John Donoghue, John Squires, Phillip De Bondi, and Wayne Piekarski. First person indoor/outdoor augmented reality application: Arquake. *Personal Ubiquitous Comput.*, 6(1):75–86, 2002.

[UMKN96] A. Utsumi, T. Miyasato, F. Kishino, and R. Nakatsu. Hand gesture recognition system using multiple cameras. In *ICPR '96: Proceedings of the 1996 International Conference on Pattern Recognition (ICPR '96) Volume I*, page 667, Washington, DC, USA, 1996. IEEE Computer Society.

[UO99] Akira Utsumi and Jun Ohya. Multiple-hand-gesture tracking using multiple cameras. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:1473, 1999.

[VB05] Daniel Vogel and Ravin Balakrishnan. Distant freehand pointing and clicking on very large, high resolution displays. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 33–42, New York, NY, USA, 2005. ACM.

[vHB01] Christian von Hardenberg and François Bérard. Bare-hand human-computer interaction. In *PUI '01: Proceedings of the 2001 workshop on Perceptive user interfaces*, pages 1–8, New York, NY, USA, 2001. ACM.

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001.

[VJ04] Paul Viola and Michael J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, 2004.

[WH01] Y. Wu and T. S. Huang. Hand modeling, analysis, and recognition. *IEEE Signal Processing Magazine*, 2001.

[WHT03] Liang Wang, Weiming Hu, and Tieniu Tan. Recent developments in human motion analysis. *Pattern Recognition*, 36:585–601, 2003.

[Wil06] Andrew D. Wilson. Robust computer vision-based detection of pinching for one and two-handed gesture input. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 255–258, New York, NY, USA, 2006. ACM.

[WKCT06] I. Wanderley, J. Kelner, N. Costa, and V. Teichrieb. A survey of interaction in mixed reality systems. In *In Symposium on Virtual Reality*, pages 1–4, 2006.

[WO90] Colin Ware and Steven Osborne. Exploration and virtual camera control in virtual three dimensional environments. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, pages 175–183, New York, NY, USA, 1990. ACM.

[WP03] Andrew Wilson and Hubert Pham. Pointing in intelligent environments with the worldcursor. In *INTERACT*, 2003.

[WS03] Jessica Junlin Wang and Sameer Singh. Video analysis of human dynamics - a survey. *Real Time Imaging*, 9:321–346, 2003.

[WWBH97] Aileen Worden, Neff Walker, Krishna Bharat, and Scott E. Hudson. Making computers easier for older adults to use: Area cursors and sticky icons. In *CHI*, pages 266–271, 1997.

[WYZ07] Stefan Winkler, Hang Yu, and ZhiYing Zhou. Tangible mixed reality desktop for digital media management. *Stereoscopic Displays and Virtual Reality Systems XIV*, 6490(1):64901S, 2007.

[Zac00] G. Zachmann. *Virtual Reality in Assembly Simulation – Collision Detection, Simulation Algorithms, and Interaction Techniques*. PhD thesis, Darmstadt University of Technology, Germany, Department of Computer Science, may 2000.

[ZFS97] Robert C. Zeleznik, Andrew S. Forsberg, and Paul S. Strauss. Two pointer input for 3d interaction. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 115–ff., New York, NY, USA, 1997. ACM.

[Zhe09] Tan Zheng. Development of interaction interfaces between 3d games and markerless hand-tracking. Master's thesis, Rheinische Friedrich-Wilhelms Universität Bonn, Germany, September 2009.

Publications

- Markus Schlattmann, Ferenc Kahlesz, Ralf Sarlette, and Reinhard Klein. Markerless 4 gestures 6 DOF real-time visual tracking of the human hand with automatic initialization. *Computer Graphics Forum*, 26(3):467–476, September 2007.
- Markus Schlattmann and Reinhard Klein. Simultaneous 4 gestures 6 DOF real-time two-hand-tracking without any markers. In *ACM Symposium on Virtual Reality Software and Technology (VRST '07)*, November 2007.
- Markus Schlattmann, Tanin Na Nakorn, and Reinhard Klein. 3D interaction techniques for 6 DOF markerless hand-tracking. In *International Conference on Computer Graphics, Visualization and Computer Vision (WSCG '09)*, February 2009.
- Markus Schlattmann and Reinhard Klein. Efficient bimanual symmetric 3D manipulation for markerless hand-tracking. In *Virtual Reality International Conference (VRIC '09)*, April 2009.
- Markus Schlattmann, Johannes Broekelschen, and Reinhard Klein. Real-time bare-hands-tracking for 3D games. In *IADIS International Conference Game and Entertainment Technologies (GET '09)*, pages 59–66. IADIS Press, June 2009.
- Markus Schlattmann and Reinhard Klein. Hybrid cursor control for precise and fast positioning without clutching. In *SIGGRAPH ASIA '09: ACM SIGGRAPH ASIA 2009 Sketches*, page 1, New York, NY, USA, December 2009. ACM.

- Markus Schlattmann and Reinhard Klein. Efficient bimanual symmetric 3D manipulation for bare-handed interaction. Accepted for publication in *Journal of Virtual Reality and Broadcasting (JVRB)*, 2010.
- Markus Schlattmann, Tan Zheng, Johannes Broekelschen, and Reinhard Klein. An investigation of bare-hands-interaction in traditional 3d game genres. Accepted for IADIS International Journal on WWW/Internet (IJWI), 2010.
- Markus Schlattmann and Reinhard Klein. Method for automatically detecting at least the type and/or location of a gesture formed using an appendage, particularly a hand gesture. Rheinische Friedrich-Wilhelms Universität Bonn, WO 2009/027307 A1, Pr.: DE102007041482A1 31.08.2007, International Patent, March 2009.
- Markus Schlattmann and Reinhard Klein. Hybrid Cursor Control. EP09-015378.4, European Patent Application, December 2009.

Awards

- Winner of the RTT Emerging Technology Contest, Markerless 6+4DOF real-time visual tracking of the human hand with automatic initialization, *8th RTT Conference*, May 10-11 in Berlin, Germany, 2007.
- Laval Virtual Award, Bare-Handed 3D Interaction, best project in the category "Interfaces and Materials", *11th Laval Virtual Conference*, April 22-26 in Laval, France, 2009.
- Outstanding Paper Award, Real-Time Bare-Hands-Tracking for 3D Games, *IADIS International Conference Game and Entertainment Technologies*, June 17-19 in Algarve, Portugal, 2009.